ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Artificial Intelligence in Industry

INTERPOLATION OF ANNUAL MAXIMUM RAINFALL PROBABILITY DISTRIBUTION USING GRAPH NEURAL NETWORKS

CANDIDATE
Luciano Massaccesi

SUPERVISOR
Prof. Michele Lombardi

CO-SUPERVISOR
Dott. Andrea Magnini

Academic year 2022-2023

Session 6th

Abstract

The modelling of high resolution maximum rainfall depth distribution plays a key role in preventing natural disasters, such as floods. The rain gauges in meteorological stations distributed over the territory already offer a good base to estimate the maximum rainfall depth distribution. However, the resolution at which the distribution is estimated can be increased through interpolation. In this thesis, the problem of maximum rainfall depth distribution interpolation is addressed by using a graph neural network-based approach. Some variations of the proposed algorithm are compared and evaluated on the northern Italy dataset. In particular, the use of a combination of a simple ensemble and temporal ensemble proved to be particularly effective. Finally, the proposed methods show improved results when compared with the classical methods: ordinary kriging, universal kriging and Gaussian process.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Maximum rainfall depth refers to the highest quantity of rainfall that can be measured over a specific duration. Given the stochastic nature of rainfall, the maximum rainfall depth varies annually and this variability can be described by the maximum rainfall depth distribution. Understanding the parameters of this distribution, referred to as maximum rainfall depth distribution parameters (MRDP), is crucial for modeling extreme rainfall events, particularly in preventing natural disasters like floods [7].

Accurate estimation of MRDP with high spatial resolution is essential for predicting the likelihood of specific disasters. Traditionally, rainfall estimates are derived from rain gauges in meteorological stations scattered across the territory. However, the limited number of stations and uneven coverage necessitates the interpolation of MRDP between gauge locations. Classical methods like ordinary kriging and Gaussian process predict MRDP based on neighboring stations and their distances, while universal kriging with external drift incorporates external factors such as altitude. While effective in highly correlated scenarios, these methods may struggle to capture complex relationships within the data.

Recent studies have highlighted the ability of Graph Neural Networks

(GNN) to learn relationships within graph-structured data. GNNs have demonstrated success in tasks such as spatial interpolation [23, 34] and rainfall forecasting [45] , suggesting that GNN can also be effective in interpolating the distribution of maximum rainfall depth. In this thesis, a transductive GNN approach for interpolating MRDP is proposed, introducing two models based on edge graph attention layer: Tiny eGAT and Extended eGAT. The two models differ mainly in the number of layers, with Tiny eGAT having fewer GNN layers than Extended eGAT. Various modifications to these models are then proposed, compared and evaluated using the northern Italy dataset.

The thesis explores the effectiveness of jumping knowledge, a variation facilitating learning in networks with multiple GNN layers. Results from the northern Italy dataset reveal indeed that jumping knowledge enhances Extended eGAT more than Tiny eGAT. Another variation involves employing a base ensemble alongside a temporal ensemble. Both ensemble techniques demonstrate improvement, with their combined application synergistically enhancing network performance. A proposed metric evaluates prediction confidence, considering multiple predictions generated by ensemble sub-models. Then, two metrics to weigh meteorological station reliability are compared and utilized to enhance the training process.

Finally, the problem of overfitting is addressed by comparing two different techniques: dropout and edge drop. While edge drop proves effective in reducing overfitting and improving results, dropout performs poorly. Overall, the proposed variations applied to Tiny eGAT and Extended eGAT models significantly enhance MRDP prediction compared to ordinary kriging, universal kriging, and Gaussian process.

Chapter 2

Related work

In the past literature, the problem of interpolating maximum rainfall distribution parameters (MRDP) using graph neural networks has not been addressed yet. However, there are many studies that have faced similar problems, which can be divided into four categories depending on the type of problem:

- Classical approaches to rainfall interpolation
- Artificial neural networks used for rainfall interpolation
- GNN used for rainfall interpolation
- Other related problems

Classical approaches to rainfall interpolation There are many different approaches to rainfall interpolation that do not use artificial neural networks (classical approaches): some of the most used ones are [21, 28]: simple kriging with varying local means, ordinary kriging, regression kriging, inverse distance weight and Thiessen polygon. Another promising classical approach is to use thin plate spline (TPS) to interpolate daily rainfall [39, 15]. TPS is a technique for smoothing a surface by minimizing its curvature. This approach obtains a fairly good estimation of rainfall when compared to other classical approaches such as isohyetal and Thiessen polygon techniques.

Artificial neural networks used for rainfall interpolation Differently from this thesis, several researches tried to interpolate rainfall using neural networks approaches that do not use graph neural networks. Rainfall rates have been estimated by an artificial neural network using both infrared satellite imagery and ground-surface information [14]. Another attempt to predict the rainfall has been made by exploiting the information of the nearby stations, using a feed forward network [20]. In recent years, transformers managed to achieve astounding results in the field of natural language processing (NLP). Inspired by its success, an attempt has been made to apply spatial transformers to address the problem of rainfall interpolation [24].

GNN used for rainfall interpolation In literature, there are many researches that use graph neural networks (GNN) to interpolate rainfall related data such as hourly rainfall. However, these researches differ from this thesis by the features that are being interpolated. One attempt has been made by applying GNN to interpolate hourly rainfall by using an adaptive graph structure learning and by constraining message passing flow [23]. Another possible approach is to use the radar reflectivity signal to model quantitative precipitation estimation. In this case a variation of graph neural network that uses the attention mechanism has been used to model the spatial-temporal relationship between radar reflectivity and quantitative precipitation estimation [34].

Other related problems While the previous researches that have been introduced are similar with the problem of MRDP interpolation, there are many other researches that tackle problems that are similar, yet more different than the previous ones. Regardless the differences, some aspects of the researches that will be described could give some ideas to tackle also the problem of MRDP interpolation. Rainfall forecasting problem has been addressed in many researches, using spatio-temporal GNN [45]. High-resolution rainfall-runoff models have been developed by utilizing high-resolution precipitation data

and geographical information through a GNN [42].

Chapter 3

Theoretical frame

In this chapter the main algorithms and techniques used in this research are described. First, the interpolation problem is introduced, together with some classical algorithm typically used in similar cases. Then, a generic definition of graph neural network (GNN) is proposed, followed by one type of GNN called edge graph attention layer. Next, a few techniques that can be used to improve the GNN performances are shown. In particular, jumping knowledge aims to improve the learning process of the network when too many layers are added. Ensembles are used to exploit the predictions multiple neural networks to obtain a better model. Eventually, two ways of estimating the distribution reliability of each data-point are described. This last estimation will be used in Section 4.4 to weigh the training samples accordingly.

3.1 Interpolation problem

Let A be a set of data-points with some associated features. Knowing the data-points A, interpolation is the problem of creating new data-points between the existing ones in A. Often, the new data points to be created have some known features and the remaining features must be determined based on the known features (Figure 3.1).

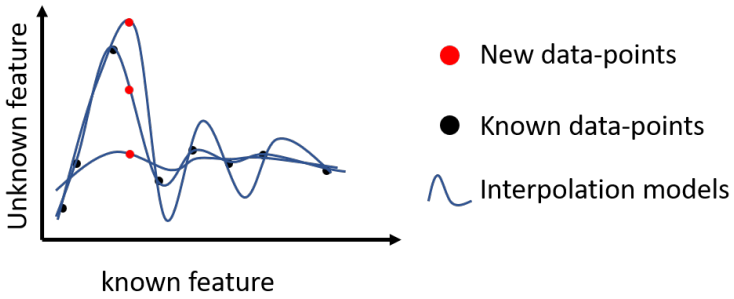


Figure 3.1: Interpolation over two features, one is known for every data point, the other is an unknown feature that must be estimated for the new points.

In the case of spatial interpolation, the data-points correspond to spatial locations and spatial closeness is often assumed to be correlated with the similarity of the features. In geostatistics, the field of statistics applied to geology, some of the most used algorithms for spatial interpolation are ordinary kriging, universal kriging and Gaussian process.

3.1.1 Ordinary kriging

Ordinary kriging interpolates the unknown features on the target data-points by a weighted average of the same features from the known data-points nearby [10]:

$$Z^*(u) = \sum_{i=1}^{N}(\lambda_i(u) * (Z(u_i) - m(u_i)) + m(u)$$

Here, $u_1, ..., u_N$ are the positions of the neighbour data-points, $Z^*(u)$ are the predicted values at the position u, $Z(u_i)$ are the known values at the position u_i and $\lambda_i(u)$ are the weights of the data-point i. The value $m(u)$ and $m(u_i)$ are the expected average of the values at the respective positions u and u_i. In ordinary kriging, the expected average of the values around the target position u are assumed to be constant. Therefore, by also imposing that $\sum_{i=1}^{N}\lambda_i(u) = 1$, the formula can be simplified as follows:

$$Z^*(u) = \sum_{i=1}^{N}\lambda_i(u) * Z(u_i)$$

The weights $\lambda_i(u)$ are determined based on a variogram model. The variogram

model is a measure of correlation based on distance between the target data-point at position u and the known data-point at position u_i. In this case, it is also assumed that the mean of the values is constant around the target station.

3.1.2 Universal kriging with external drift

While ordinary kriging assumes stationary data-points around the target position (the average values around the target position $m(u)$ are constant), universal kriging with external drift assumes that there is no such property [10]. Instead, the average values around the target position depend on an external function. By adding the external drift, universal kriging could obtain better predictions. However, a strong correlation between the external drift function and the values that must be predicted is necessary.

3.1.3 Gaussian process

Gaussian process assumes that the data follows a multivariate normal distribution and fits this distribution on the known data. Once this is done, the known distribution can be used to find the values on new data-points [36]. Multivariate normal distribution is defined by the following probability density function [1]:

$$m.n.d. = \frac{1}{\sqrt{(2\pi)^d det(\Sigma)}} exp\Big(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\Big)$$

Where the data has k dimensions, $det(\Sigma)$ is the determinant of Σ and T stands for transpose. This formula depends only on μ, which is the expected average and Σ which is the covariance matrix. The expected average can be set to zero by shifting all the data. This leaves only Σ to be defined.

The covariance matrix Σ of the N data-points can be defined as:

$$\Sigma = \begin{pmatrix} k(1,1) & ... & k(1,N) \\ ... & ... & ... \\ k(N,1) & ... & k(N,N) \end{pmatrix}$$

Where $k(i, j)$ is the covariance between the data-points i and j. The covariance $k(i, j)$ is then defined by a function called kernel which is often a parametric function that depends on some parameters. Those parameters are then optimized to maximize the likelihood that the distribution represents the known data.

3.2 Graph Neural Networks

This section aims to define everything needed to understand graph neural network (GNN). As the name suggests, GNN are artificial neural networks applied to graphs. Therefore, graphs and artificial neural networks are defined first. Then, a definition for graph neural networks is proposed. As more variations of GNN have been proposed through history, the definition of GNN has been extended to include the new methods. This section will focus on a general definition of GNN that includes as many GNN variations as possible, including the variation used in this thesis.

3.2.1 Graphs

A graph is a data structure composed by nodes and edges as shown in Figure 3.2. Nodes can contain some data (node features), while edges are entities that connect two nodes and can also contain some data (edge features) that represents the relationship between the nodes. Additionally, more inclusive definitions of graph also consider the possibility of having some additional data related to the graph called global features [6]. More formally, a graph G is defined as $G = (V, E, u)$ where V is the set of all nodes, E is the set of edges and u are the global features.

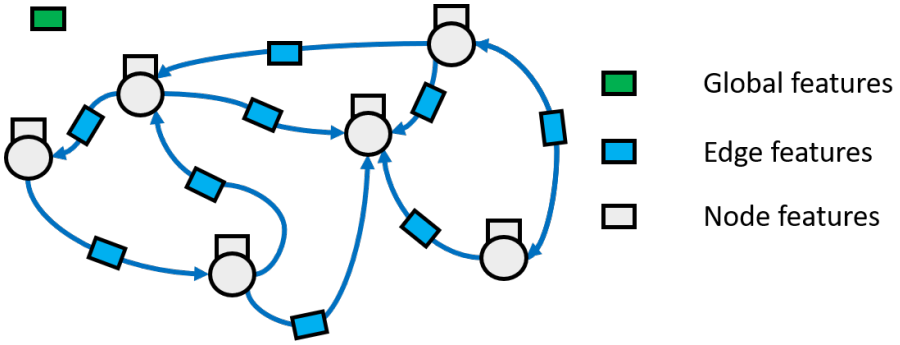


Figure 3.2: Graph scheme

3.2.2 Artificial neural networks

One of the first models for neural networks was proposed with the idea of simulating neurons in the human brain [30]. In recent years, artificial neural networks (ANNs) have been applied successfully, surpassing the state of the art in many different fields. Because of its wide usage, there are many papers that summarize or explain what artificial neural networks are [19, 2]. The base idea behind artificial neural networks is to simulate how the brain works. In particular, by reproducing the functionalities of the basic entities in our brain called neurons, ANNs aim to reproduce complex functions through learning from experience.

Artificial neurons

Neurons are entities in human brain that can have some input signals and based on that, process an output signal [17]. Some of them interact through the environment by receiving external signals (such as sight, smell, touch, ...) and by moving the muscles in the body. Other neurons are connected together to elaborate the information received by the input neurons and propagate the result all the way to the output neurons where they must be used. In similar ways, artificial neurons are the basic entities of the artificial neural network. Artificial neurons can either receive an input from the environment or can be connected to the output of many other neurons. Those input signals are then

modified and aggregated to generate an output (Figure 3.3). Given a neuron x_i and a set of neurons $x_1, ..., x_N$ directly connected to x_i, the value of the neuron can be computed as follows [19]:

$$x_i = f(\sum_{j=1}^{N}(w_{j,i} * x_j) + b_i)$$

Here f is called activation function and is usually needed to add non linearity. $\sum$ in a more general definition of ANN can be substituted by a different aggregation function. $w_{j,i}$ are the weights of the edges that connect the node with its neighbours. b_i is called bias and can also be seen as a weight of a constant input ($b_i * 1$). The weights ($w_{j,i}$ and b_i) are values that can be modified to change the output of the artificial neural network, encoded in the output layer. This process is called training.

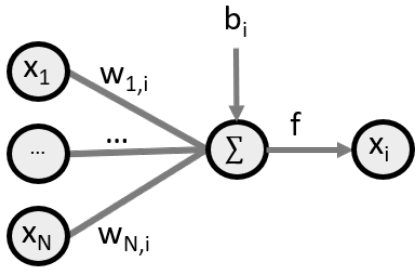


Figure 3.3: Artificial neuron scheme.

Network of artificial neurons

Artificial neural networks (ANNs) are graphs composed by artificial neurons connected together through directed edges. Similar to the brain, a set of those neurons is commonly called input layer and accepts an external input. Another set of neurons is called output layer and its value is considered to be the output of the artificial neural network. The other neurons are also commonly structured in sequences of layers, called hidden layers. In every layer neurons are usually connected to the neurons of adjacent layers, but in some cases they can also be connected to neurons in distant layers. Figure 3.4 represents the scheme of a basic ANN. Here $X_1, ..., X_3$ are the inputs of the ANN and are also the values of the neurons in the input layer. $Y_1, ..., Y_3$ are the outputs of

the ANN and correspond to the values of the output layer.

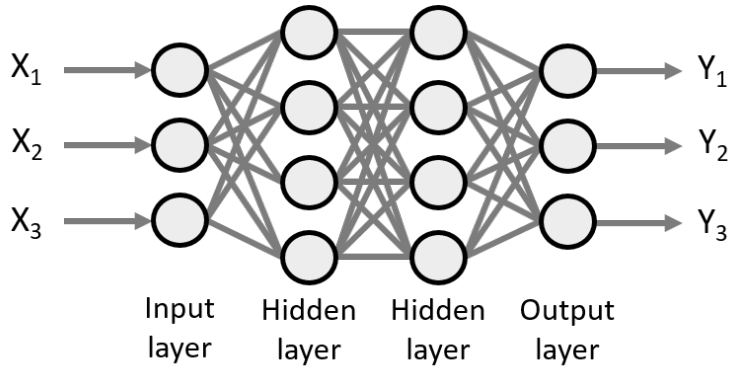


Figure 3.4: Artificial neural network scheme.

In the field of neural networks, there are some common terms that are often used to define particular structures or elements of the neural networks. A dense layer is a layer of artificial neurons, each of them fully connected with the previous layer (such as the hidden layers in Figure 3.4). A feed forward network is a neural network without any cycles. When more complex network structures are used (such as graph neural network layers), the term hidden units is defined as the number of artificial neurons used internally.

Non linearity in ANN

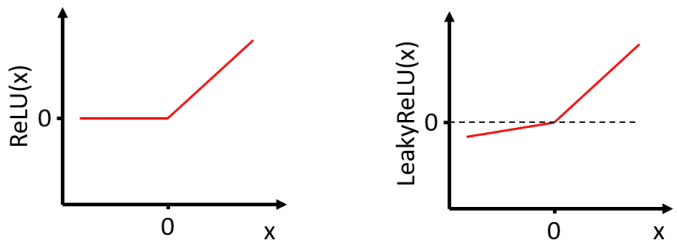
Every neuron in the ANN can be seen as a function that, taken some input, gives an output. In the same way, the ANN can be seen as a function composed by many other functions (artificial neurons). However, if the artificial neurons are linear functions, the ANN will also be a linear function. This would limit the complexity of problems that the ANN could solve [37]. In order to fully exploit the potential of ANNs it is necessary to properly add the activation function to the artificial neurons and use it to add non-linearity. There are many possible activation functions, each of them with some application fields.

One of the most used is the ReLU defined as $ReLU(x) = max(0, x)$ (Figure 3.5a). Despite its simplicity, this function adds non linearity and has proven to perform well in a large variety of situations. However, either during

artificial neuron's weight initialization or during training, it can happen that the output of the neuron before the activation function is less than zero. In this case, the output of the activation function is zero and the derivative of the function is also zero. As most of the optimizers rely on the derivative of the artificial neurons to change the weights, the optimizers could get stuck with the same weights. When this situation occurs for every possible input in the training set, the neuron is said to be dead and might not recover from that situation. This phenomena is known as dying ReLU [25]. While some dead neurons are not a problem, if the number of dead neurons increases, the performance of the ANN could be limited. To solve this problem, a variation of ReLU called LeakyReLU [26] is proposed (Figure 3.5b):

$$LeakyReLU(x) = \begin{cases} \beta * x & x < 0 \\ x & x >= 0 \end{cases}$$

Where β is a positive small number. This function is similar to ReLU in its functionalities but avoids the problem of dying ReLU.



(a) ReLU function. (b) LeakyReLU function.

Figure 3.5

3.2.3 Learning process

The learning process of ANNs can be divided into three steps:

- Training phase
- Validation phase
- Test phase

Training phase Similar to real brains, ANNs are capable of learning from experience by changing the weight of the connections between neurons. This process is called training phase. In this phase, the ANN is applied to the training set and the weights of each neuron are optimized to minimize (or maximize) a loss function. The loss function is a function that, given the input, the output and the weights of the ANN, returns a score of how bad (or good) the ANN is performing.

Validation and test phases During the validation and test phases, the network is tested on datasets different from the one used for training. While the test phase is used to test and evaluate the ANN, the validation phase is used to compare different hyper-parameters and variations before the test phase. Because of the fact that the hyper-parameter and variations are chosen knowing the dataset used for validation, the dataset used for test is usually a different one. The addition of the validation step is useful to avoid adapting the ANN knowing the dataset used for testing, which would make the results unrealistic.

Transductive learning When the validation and test set are considered to be completely unknown during the training phase, the learning process is called inductive learning. When some features of the validation and test sets are known during the training phase, the learning process is called transductive learning. In this last case, however, only a part of the features in the validation and test set is known during training. The target features, which are the ones that must be predicted, are not known during the training phase. If they were known, there would not be the need to build any predictive model in the first place. It is therefore important that the target features from the validation and test sets are kept hidden from the ANN during training phase. As an example, one can consider the problem of predicting the traffic jam in certain areas of a city. In this case, the number of cars is only known for some roads, while unknown for others. However, the position and shape of all the roads is well

known, even in the training phase.

3.2.4 Graph neural networks

Graph neural network (GNN) is a class of ANN that works on graph structured data and aims to produce new features for the nodes of the graph, the edges or general features regarding the graph as a whole. The first formal definition was proposed as an extension of recurrent neural networks [11]. More recently, the framework of message passing to propagate the information through the graph has been consolidated [9]. Message passing is based on the idea of passing pieces of information, called messages, between neighbour nodes of the graph for each iteration. Each time the message is passed between the nodes, new node features are generated based on: the features in the sending nodes, the features in the receiving node, the features of the edges that connects the nodes and the global features. A brief scheme of message passing is shown in Figure 3.6. The message passing framework has been further extended to include more types of models [6]. In this final formulation, at each step the value of a node is updated as follows:

$$h'_j = \Phi(h_j, u, A_{i \in N}(\Psi(h_i, X_j, e_{i,j}, u)))$$

Where:

- h_j are the node j features
- h'_j are the new node features
- N is the set of nodes adjacent to h_j
- $e_{i,j}$ are the edge features from h_i to h_j
- u are the global features
- A is an aggregation function and must be permutation invariant (often simple functions are used, such as min, max or sum)
- Φ and Ψ are typically trainable functions

Usually, this process is repeated multiple times, either by using the same instance of this function or by stacking more different layers (often referred as GNN layers), one after the other.

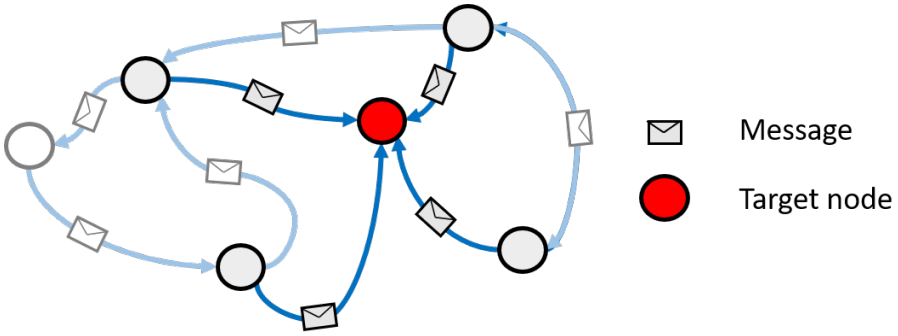


Figure 3.6: GNN message passing scheme. In this case the node features of the target node are being updated through messages passed by its neighbours. The messages represent the information passed from the neighbour nodes.

3.2.5 Overfit, underfit, dropout

In machine learning, the aim is often to create a model that approximates a certain expected behaviour. When data is split into training, validation and test sets as explained in Section 3.2.3, the training set represents the known expected behaviour, while validation and test sets must be predicted and are assumed to be similar to the training set, but not identical. As the model is created by using the training set only, two known phenomena can happen, known as overfit and underfit (Figure 3.7).

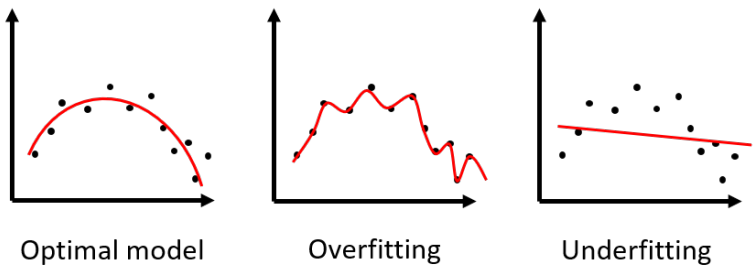


Figure 3.7: Overfitting and underfitting of a simple dataset.

Overfit The real data often contains random noise. When a model focuses on predicting the noise correctly, instead of generalizing the behaviour of the data, the overfitting problem occurs [44]. In practice, the model is said to be overfitting when it correctly predicts the data that has been used to create that model, while it wrongly predicts the data that has not been seen before. This often occurs when the model correctly predicts the training set, while it wrongly predicts the validation and test sets. However, as the model also partially depends on the validation set, it could also happen that the model performs well on training and validation sets, while it performs poorly on the test set.

Underfit Underfitting is the opposite problem of overfitting. In this case, the model generalizes too much. When this occurs, the model also wrongly predicts the same data that has been used for training. In order to avoid both overfitting and underfitting, it is necessary to find a compromise between generalizing too little and too much. When using ANN, it can be challenging to manually calibrate the structure of the ANN and the learning process to avoid both problems. However, here are some general techniques that can help to prevent overfitting while avoiding the underfitting case. One commonly used technique is the dropout.

Dropout Dropout is a technique used in ANN that consists in randomly removing some connections between neurons during every training step (Figure 3.8). One way of motivating the dropout is to see every combination of the network with removed connections as a different, smaller network [38]. For each training step, one of those smaller networks is taken and trained. At the end of the training phase, the resulting network is an average of all the smaller networks. By averaging multiple models, the result is more generalized.

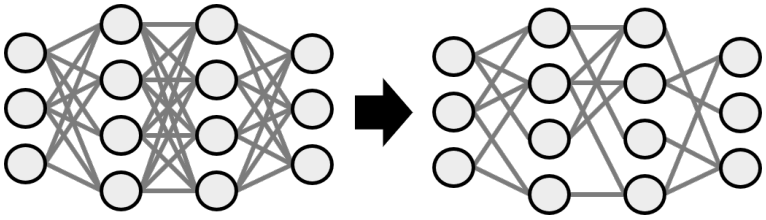


Figure 3.8: Dropout technique example applied to a simple ANN.

Edge drop When the data in the training set is not sufficient, the network has difficulties to generalize, leading to the problem of overfit. This could be solved by increasing the size of the dataset however, this is often not possible. One possible solution is to artificially create more data by modifying the available dataset. In the case of GNN, this can be done by randomly removing some edges for each training epoch [35]. By doing so, the network has to learn every time on a slightly different graph and therefore is forced to generalize more. This technique is similar to dropout, in the sense that both techniques remove edges from graphs. Nonetheless, the idea behind them is completely different: dropout changes the structure of the neural network, edge drop augments the data.

3.3 Attention

Attention is a mechanism that is used to focus the resources available on a subset of the input features that are more relevant to the final goal. As for many other artificial intelligence techniques, this one was inspired by nature. The idea of attention has its roots in the attention mechanism in human brains. In this analogy, the neural network is seen as a brain and has limited resources. By focusing the brain's resources, humans can achieve great performance in tasks such as finding relevant objects using our visual system [8]. In the similar way, neural networks can use attention to achieve better results.

3.3.1 Attention in artificial neural networks

In one of the main reference models, attention is applied to the problem of machine translation [4]. Similarly to human brains, the attention mechanism scores how much a certain input is important given a certain context. The input is then weighed according to its relevance. More formally, at every time step i, the input of the neural network is replaced by a context vector which is defined as follows:

$$context\_vector_i = \sum_{j=1}^{N} \alpha_{i,j} h_j$$
$$\alpha_{i,j} = softmax_j(a(S_{i-1}, h_j))$$
$$softmax_j(a(S_{i-1}, h_j)) = \frac{exp(a(S_{i-1}, hj))}{\sum_{k=1}^{N} exp(a(S_{i-1}, h_k))}$$

Where:

- $h_1..h_N$ is a list of N input.
- S_i is a state variable at time i, which is used to determine the importance of the inputs. This variable can be seen as the context in which the attention must be applied.
- a is a function that must determine how much a given input h_1 is important. In [4] a feed forward network is used.

In the case of multi-head attention, the attention mechanism is repeated multiple times in parallel to obtain more final $context\_vectors$ instead of one [40]. These multiple $context\_vectors$ are usually aggregated by an aggregation function such as concatenation or sum.

3.3.2 Graph attention layer

The concept of attention can also be used in more elaborated network structures, such as graph neural networks. The main idea is to weigh the information received from every neighbour node based on their relevance. A graph

attention layer (GAT) is defined as a GNN layer that uses the attention mechanism. one possible definition of GAT is the following [41]:

$$h'_j = \sigma(\sum_{i \in N} \alpha_{i,j} W h_j)$$
$$\alpha_{i,j} = softmax_j(LeakyReLU(a^T(Wh_i || Wh_j))$$

Where:

- h'_j are the new features of node j
- h_j are the current features of node j
- σ is a non linear function (usually referred as activation function)
- W are the trainable weights
- a^T is a transposed trainable weight vector
- $||$ is the concatenation function

It is then possible to stack in parallel more instances of this GAT layer to obtain a multi-head GAT layer.

3.3.3 Edge graph attention layer

The definition of GAT explained in Section 3.3.2 only uses neighbour nodes features to compute the new node features. However, edge features can be crucial to evaluate the relevance of neighbour nodes. As an example, when the nodes of the graph represent spatial location, the edges can contain information about the distance between nodes. In many different real cases, spatially close points are more correlated than distant points. Therefore, the distance between the neighbour nodes can be an indicator of how much the neighbour node is relevant to the target node. GAT can be modified to also includes edge features, in such case it is called edge GAT (or eGAT) [32]. Formally, the new node features h'_j are computed as follows:

$$h'_j = W_s * h_j + \sum_{i \in N} \alpha_{i,j} W_n * h_j + W_e * e_{i,j}$$

$$\alpha_{i,j} = softmax_j(LeakyReLU(a^T(W_n * h_i || W_n * h_j || W_e * e_{i,j}))$$

Where W_s, W_n and W_e are trainable weights and $e_{i,j}$ are the edge features of the edge that connects node i to node j.

3.4 Jumping knowledge

Jumping knowledge is a technique used to make the learning process easier when the network is deep (i.e. it has a high number of layers). This technique can be seen as an extension of a simpler one called residuals, so in this section residuals will be defined first.

3.4.1 Residuals

When designing neural networks, adding more layers can create more complex functions and by consequence can lead to better solutions. However, the more layers there are, the more difficult it is to train the network. This problem is particularly evident with convolution neural networks or similar neural network structures such as graph neural networks. This problem has been addressed in several researches and different solutions have been found to try to mitigate this problem. One solution that has proven to be effective in many different real cases is the use of residual connections [12]. Given a neural network layer (which could also be a graph neural network layer), the residual connection consists in summing the input of the layer to its output. More formally, given an input X and a layer $F(X)$, the output of the residual connection will be $X + F(X)$. This value is usually used as a substitution of the raw output of the layer $F(X)$ (Figure 3.9). This solution has proven to improve the results for networks with a high number of layers. After that, many other similar solutions have been proposed. One of them is known as jumping knowledge.

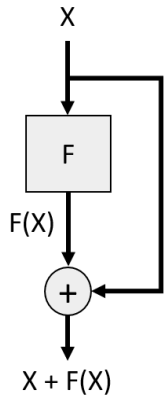


Figure 3.9: Residuals scheme. The general network layer is represented by the function F, X is the input and x + F(X) is the output of the residual

3.4.2 Jumping knowledge

Similar to residuals, jumping knowledge also creates connections that skip neural network layers. However, differently from residuals, jumping knowledge is able to create connections between a layer and all previous layers [43]. As a particular case of neural network, jumping knowledge can also be applied to GNN. In such case, jumping knowledge can be applied node by node independently. More formally, given a node and its features in N previous layers $h_1, ..., h_N$, the result of the jumping knowledge is an aggregation of such features. The core part of this technique is how the features $h_1, ..., h_N$ are aggregated. Three possible aggregation functions are suggested:

- concatenation: $(h_1||...||h_N)$
- max pooling: element-wise $max(h_1, ..., h_N)$
- LSTM-attention: a bi-LSTM is used on $h_1, ..., h_N$ to obtain the attention weights $a_1, ..., a_N$. The attention weights are then used to aggregate $h_1, .., h_N$ as follows: $\sum_{i=1}^{N} h_i * a_i$

LSTM is a type of recurrent neural network that can process sequentially a list of input $h_1, ..., h_N$ and remember useful information of the previously

visited part of the input in its hidden state [13]. For every input h_i, an output is generated and the hidden state is modified. A bi-SLTM is a union of two LSTMs where the first one receives the input in order $h_1, ..., h_N$, while the second one receives the same input but in reverse order $h_N, ..., h_1$. The output of the two LSTMs is then concatenated.

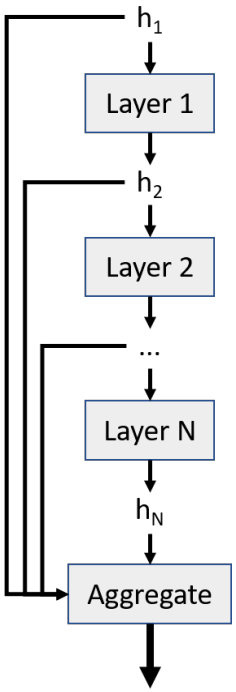


Figure 3.10: Jumping knowledge scheme.

3.5 Ensemble

In machine learning regression problems, ensemble is defined as a technique that combines the results of many different models to obtain a better prediction. This process can be divided into three steps [31]:

- Generation phase
- Pruning phase
- Integration phase

After defining these three phases, one particular ensemble model, called temporal ensemble, will be described.

3.5.1 Generation phase

In this phase, a set of models is generated and trained. This set is homogeneous when all models use the same learning process, for example many different instances of the same ANN but with different weights. Differently, this set is said to be heterogeneous when the models use different algorithms which means, in the case of ANN, different network structures or different learning processes.

3.5.2 Pruning phase

The pruning consists in removing some of the models with the aim of improving the predictions or reducing the learning or prediction time. One of the commonly used approaches is to somehow try to identify the models that are redundant and have little impact on the combined result.

3.5.3 Integration phase

After the models are trained and pruned, the models are somehow combined. In regression problems, this is usually done through a weighted sum of the models. Given an input x and a set of N models $f_1(x), ..., f_N(x)$, the output of the ensemble can be obtained as:

$$ensemble(x) = \sum_{i=1}^{N} h_i(x) * f_i(x)$$

The definition of $h_i(x)$ is then the core of the integration phase, that defines how the models are combined.

3.5.4 Temporal ensemble

Self-ensembling is a class of ensemble algorithms that does not require different models, but exploits only one model to obtain different predictions that are then aggregated. In this case, every different prediction is considered as a sub-model. The key part of self-ensemble techniques is to somehow obtain different but still good predictions from the same input, while using only one model. If this problem is solved, the great advantage is the reduction in time consumption as no more than one model has to be trained. Hence, a huge number of sub-models can be aggregated with little cost in time, contrary to other types of ensemble.

Temporal ensemble is a type of self-ensemble that uses only one trained model but at different points in time [22]. Assuming that the model used is trainable and changes over time, by considering the model at different points in time, the prediction will be different. By then aggregating those different predictions, a better result can be obtained. This process only requires to store and load either previous predictions or the status of the model at previous times. Usually, the time to load and store this information is far lower than the time needed to fully train a new model. Thanks to that, a high number of models can be aggregated with little overhead time cost.

3.6 Distribution reliability

This section aims to show different methods that can be used to score how much a certain distribution represents a certain set of data. The base idea is to calculate the cumulative distribution function (CDF) of both the real data and the estimated distribution. These methods differ on how the similarity between the two CDFs is estimated. Two well known methods are: Cramér–von Mises and Kolmogorov-Smirnov test.

3.6.1 Cumulative distribution function estimation

Cumulative distribution function (CDF) gives the probability that a certain random variable X is smaller than the variable x. More formally,

$$CDF(x) = Probability(X < x).$$

For known distribution functions, the CDF is known across the entire domain. In the case of real data, given the dataset composed by the values $X_1, ..., X_N$, the probability of having X_i is considered to be equal to $\frac{M}{N}$ where M is the number of data points in $X_1, ..., X_N$ that are exactly equal to X_i. The probability of every possible value that is not in $X_1, ..., X_N$ is equal to zero. To calculate the CDF, the resulting function will be stepwise as shown in Figure 3.11

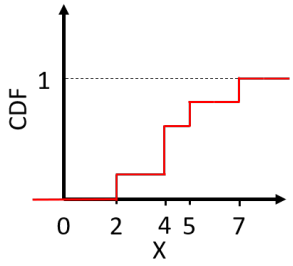


Figure 3.11: CDF function, given the real data points with values $\{1, 4, 4, 5, 7\}$

3.6.2 Cramér–von Mises and Kolmogorov-Smirnov

Given the observed values $x_1, ..., x_N$ in increasing order, Cramér–von Mises statistic is calculated as [3]:

$$\frac{1}{12N} + \sum_{i=1}^{N} [\frac{2i-1}{2N} - F(x_i)]^2$$

where $F(x_i)$ is the real distribution. The main idea is to use the squared difference between the two CDFs as a measure of error. Assuming that the values $x_1, ..., x_N$ have all the same probability, the term $\frac{2i-1}{2N}$ is the CDF averaged between the observed value x_i and the previous observed value.

Kolmogorov-Smirnov statistic is defined as [16]:

$$sup_x|F_n(x) - F(x)|$$

where sup is the upper bound of $|F_n(x) - F(x)|$, $F_n(x)$ are the real samples and $F(x)$ is the expected distribution. Intuitively, in Kolmogorov-Smirnov the statistic is the maximum distance between the CDF of the real data and the CDF of the expected distribution.

Chapter 4

Methodology

In this chapter, the problem of interpolating maximum rainfall depth distribution parameters (MRDP) is addressed by proposing two models based on edge graph attention layer (eGAT). First, a set of possible additional features is defined. These features cannot all be retrieved from existing datasets, therefore a preprocessing step is necessary. Then, starting from the previously defined features, a graph is created. In particular, it is necessary to define the nodes of the graph and the connections between them (edges), together with the features associated. The structure of the network is then presented, together with two variations called Tiny eGAT and Extended eGAT. Eventually, as the network is trained in a transductive way, particular attention will be posed to the learning process.

4.1 Preprocessing

During the preprocessing phase, MRDP is first estimated from the yearly maximum rainfall. Then, the features needed to predict MRDP are processed, starting from the available raw data. As described in Section 3.2.1, there are three types of features that can be preprocessed: global features, node features and edge features. Even though some features could be interpreted as global features (such as the hyper-parameters), the proposed algorithms do not aim to

generalize over multiple graphs. For this reason there is no need to use global features to distinguish between features that are different from one graph to another and those that are shared between different graphs. Node features and edge features must be carefully chosen and preprocessed accordingly.

4.1.1 Maximum rainfall distribution parameters

Starting from the maximum rainfall retrieved from the real stations year by year, the parameters of a distribution can be estimated. First, the type of distribution must be assessed to correctly represent the real data. The choice of the distribution is crucial; choosing the wrong distribution can indeed lead to the creation of distribution parameters difficult to predict. Every station has the yearly maximum rainfall distributed differently, so a distribution that perfectly fits all the data can only be obtained by using a more general distribution. However, the simpler the distribution is, the easier it is to predict its parameters using a neural network. For this reason, a compromise must be found on the distribution choice. This inevitably leaves some stations badly represented. A measure of how much the distribution fits the data can be estimated for every station using Kolmogorov-Smirnov or Cramér–von Mises statistics as described in Section 3.6. The Kolmogorov-Smirnov or Cramér–von Mises statistics give an error which indicates how much the real data differs from the expected distribution. Therefore, in order to obtain an actual goodness of fit measure, the value is inverted. Let R_i be the Kolmogorov-Smirnov or Cramér–von Mises statistics for the MRDP at the station i. The goodness of fit measure at that position will be $\frac{1}{R_i}$.

4.1.2 Node features

In order to predict the MRDP of the target stations, the MRDP of known stations can be used. Many interpolation algorithms can obtain fairly good results

using MRDP alone. However, the bigger is the gap between the known stations, the more it becomes difficult to predict MRDP without other knowledge about the territory. To interpolate in a more precise way, some information about the environment is required. This leads to two different type of features: known MRDP and the environmental features. While the MRDP is known in some stations and it is unknown in all other places, environmental features can be obtained everywhere needed. There are several existing datasets that contain this type of features and many more can be obtained by preprocessing the initial features. Adding non necessary features can make the overall process heavier and lead to overfit. A good selection of these features is essential for the network to correctly interpolate MRDP. The choice of these features depends on the geographical area where the MRDP must be predicted. The full list of these features used in this thesis, together with the dataset used, will be shown in Section 5.1.

4.1.3 Edge features

In the problem of interpolation, knowing how adjacent nodes are correlated can be used by the network to properly focus on the most important adjacent nodes. For this purpose, one of the best features is the distance between the nodes, as it is likely that spatially close stations have similar MRDP. To reinforce that claim it is worth noting that most of the classical algorithms for interpolation (such as ordinary kriging) use distance as a measure of weight.

4.2 Graph creation

In order to apply GNN algorithm to predict MRDP, a graph must firstly be created starting from the data. As described in Section 3.2.1, the main entities of the graph that must be defined are the nodes, the edges and global features. As suggested in Section 4.1, there are no global features, while nodes and edges are defined in the following subsections.

4.2.1 Graph nodes

Nodes are identified by their coordinates (latitude and longitude) and have features that can be divided into two groups. The first group includes the environmental features that are defined in Section 5.1, which are known for every pair of coordinates (even at the position where the meteorological stations are not present). The second group includes the parameters of the maximum rainfall distribution (MRDP), that must be predicted and are known only for some nodes.

The number and position of the nodes are in part defined by the presence of meteorological stations over the territory. For every meteorological station a node is created, regardless on whether the station is used for training, validation or test. However, the distribution over the territory of such nodes can be non uniform and therefore can lead to low density areas with very few nodes. Where the density of nodes is too low, it becomes difficult to propagate the information through the graph as little knowledge about the environment or the MRDP is contained inside it. While the initial knowledge about the MRDP is bound by the presence of real meteorological stations, having a uniformly distributed knowledge of the environment can mitigate the lack of stations. This can be achieved by creating a regular grid of artificial nodes. To sum up, there are three types of nodes:

- Known stations: stations for which both environmental features and rainfall distribution parameters are known
- Unknown stations: stations for which only environmental features are known and the rainfall distribution parameters must be predicted
- Artificial stations: support stations containing only environmental features and for which the rainfall distribution parameters are irrelevant

In addition to the features cited above, whether the MRDP is known or not can be considered a feature itself, called masking value. For implementation

purposes it is easier to always assign the MRDP to the stations, even when the MRDP is not known. In such case, the MRDP is defined as a vector of zeros. The masking value is then used to discriminate whether to consider the MRDP values or not. Figure 4.1 shows a graphical representation of the different type of nodes and their feature composition.

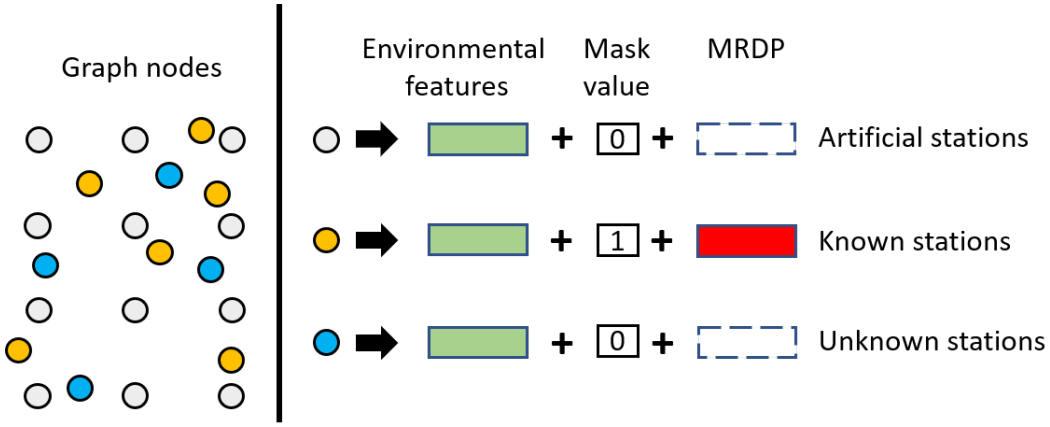


Figure 4.1: Different types of graph nodes and their feature structure. Mask value zero indicates the fact that the MRDP contains a vector of zeros and therefore must be ignored.

4.2.2 Graph edges

Once the nodes are defined, the connections between them must be created. As distant nodes are less likely to influence each other, the connections are created only between neighbour nodes. Therefore, this process depends on the definition of neighbourhood. One simple way of defining it is to set a threshold: maximum neighbour distance. All nodes closer to that distance are considered neighbours. This solution is particularly well fitted for this problem because by correctly setting that threshold, it is guaranteed that every non artificial station is connected to at least one artificial station, and every artificial station is connected to the other adjacent artificial stations. This will facilitate a regular propagation of the information through the graph and guarantee that the graph is fully connected. More precisely, let the artificial stations be created on a grid with a distance between the stations of d. The diagonal distance of the

nodes on the grid will be $\sqrt{2} * d$ as shown in Figure 4.2. By setting the maximum neighbour distance bigger than $\sqrt{2} * d$, the requirements described above are guaranteed. Also, practical tests proved that by having a bigger maximum neighbour distance the results are not improved. Therefore, A good value for the threshold can be set as slightly bigger than $\sqrt{2} * d$.

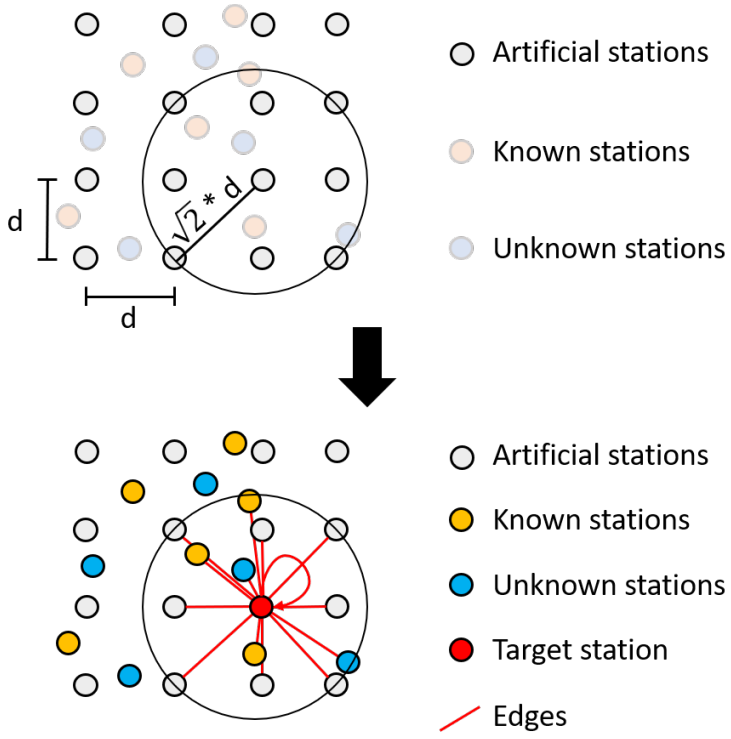


Figure 4.2: Edge creation by setting a threshold $\sqrt{2} * d$ where d is the spacing for the artificial station's grid.

When updating the features of a node, first the information from neighbour nodes is aggregated. Then, the features of the node are updated based on the result of the previous aggregation. Depending on the actual algorithm used, it is possible that when a node is updated, its previous features are not directly maintained. In such case, some useful information could be lost. This case is avoided by adding self connecting edges for every node in the graph as shown in Figure 4.2. This way, the features of the node that is being updated are aggregated in the aggregation step together with the information arriving from neighbouring nodes. Hence, the previous features of the node will be kept only

if they are still relevant, even after all information from neighbouring nodes is considered.

4.3 Network structure

The overall structure of the neural network used can be divided in three subsequent parts as shown in Figure 4.3:

- feature processing
- graph propagation
- rainfall distribution reconstruction

In the graph propagation two variations are proposed. Both of them are based on edge graph attention layers (eGAT) but while the first one has less eGAT layers with residuals, the second one has more eGAT layers uses the jumping knowledge technique. Therefore, those two version will be referred to as Tiny eGAT and Extended eGAT.

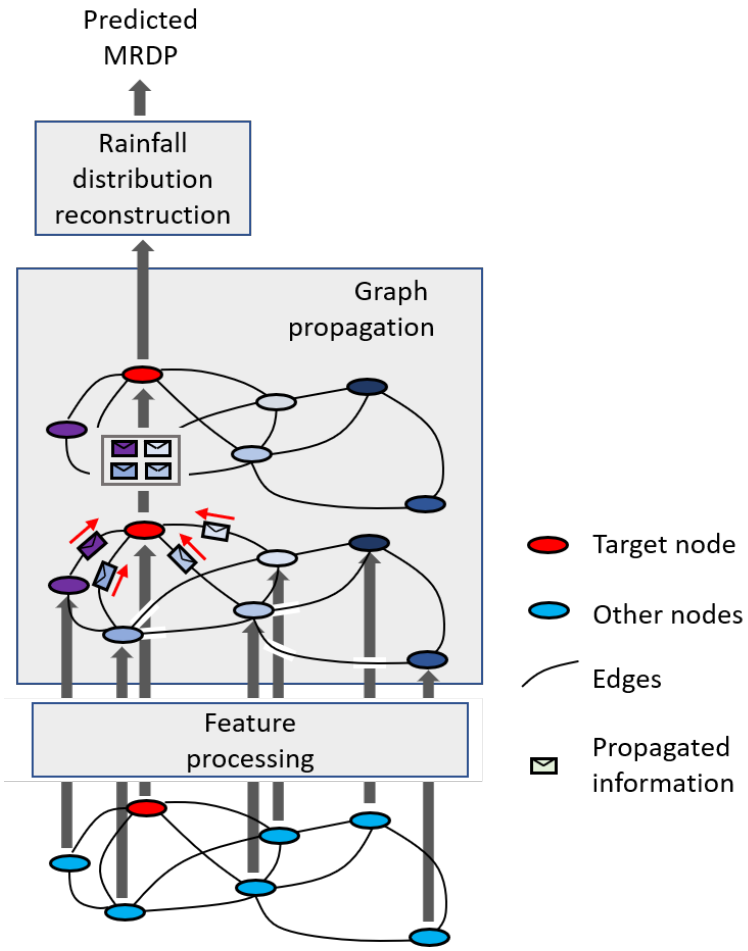


Figure 4.3: Overall network structure divided in: feature processing, graph propagation and rainfall distribution reconstruction.

4.3.1 Feature processing

The first layers of the network takes as input the preprocessed normalized data. Even if useful features have already been preprocessed, it is difficult to manually tune and create all the possible features that are required in the graph propagation step. For this reason, the feature processing layers aim at generating new features and obtain a representation of the data that is more fitted to be used by the next layers. To achieve this, a few dense layers are used, each one of them followed by activation function LeakyReLU and dropout. In practice, three fully connected layers are used with 64 neurons each. It is important to consider that in order to apply these layers to all the nodes of the

graph, there are no multiple copies of the same layer with different weights: all the nodes are processed by the same instance of the layers.

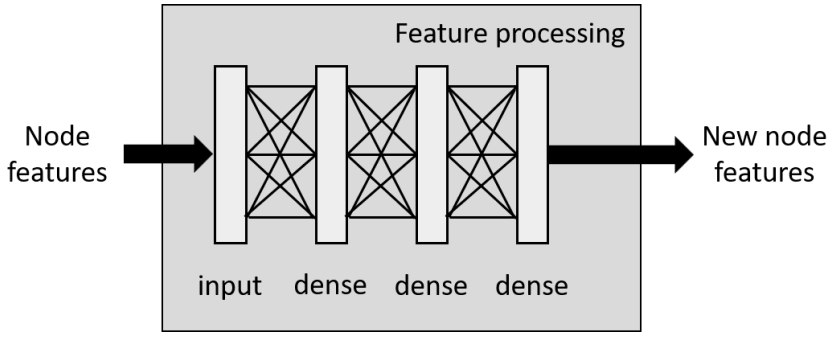


Figure 4.4: Feature processing block, takes as input node features to generate new node features.

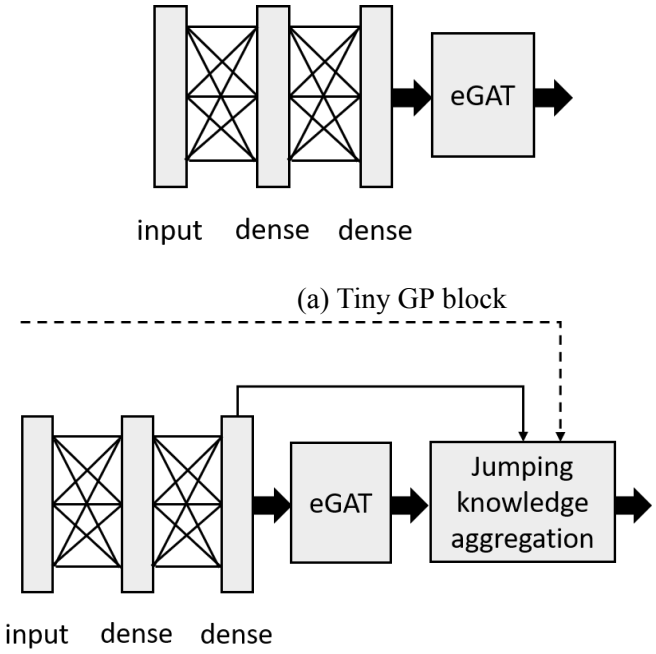
4.3.2 Graph propagation

The core part of the network is graph propagation. Here the information contained in the individual stations is propagated to the neighbouring nodes multiple times. This part is fundamental in order for useful information to reach distant nodes where the parameters of rainfall distribution must be assessed. To do so, several layers of graph propagation block (GP block) are used.

Each GP block is composed by different parts. First there is a small shallow network composed by two dense layers, each of which is then followed by the activation function LeakyReLU and a dropout. Then, an edge graph attention layer is added with five attention heads, as described in Section 3.3.3. This last layer also includes dropout and LeakyReLU. In the Tiny eGAT version, residuals are then added to the edge graph attention layer. Each dense layer is composed by 64 neurons and the graph attention network also has 64 hidden units.

In the Extended eGAT version, jumping knowledge is then added to the GP block. In this case, for each node of the graph, the features from all previous GP blocks are retrieved and aggregated. The aggregation phase is done by

first applying the following functions as described in Section 3.4.2: concatenation, max-pooling and LSTM-attention. Then, the result of the aggregation functions is also aggregated together by concatenation. The result is eventually concatenated with the output of the edge graph attention layer.

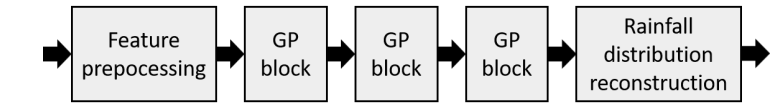


(a) Tiny GP block

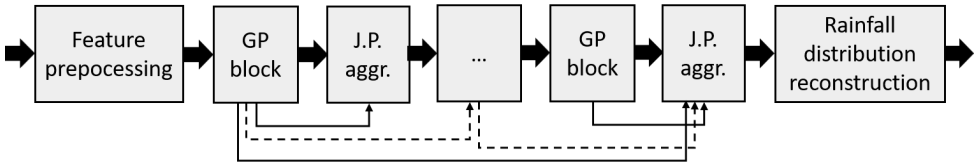
(b) Extended GP block

Figure 4.5: Tiny 4.5a and Extended 4.5b GP block representation. The dense layer and jumping knowledge aggregation is applied node by node. eGAT is applied to the whole graph by propagating the feature of each node to adjacent nodes.

The addition of jumping knowledge to this part of the network in the Extended eGAT variation makes the learning process easier when the number of GP blocks is higher. For this reason the Tiny eGAT version is limited to three GP blocks, while Extended eGAT can have 5 GP blocks. In Figures 4.6a and 4.6b the overall structure of Tiny eGAT and Extended eGAT are shown.



(a) Tiny eGAT overall structure



(b) Extended eGAT overall structure

Figure 4.6: Tiny 4.6a and Extended 4.6b eGAT overall structure. J.K. Aggr. stands for jumping knowledge aggregation.

In a base implementation of these steps, all the nodes propagate their information to their neighbours. However, knowing which nodes are the target nodes for which MRDP must be predicted, something more efficient can be done. Considering a GP block with N other GP blocks after that, the information of the nodes can be aggregated only on the nodes that are N steps far from the target nodes. All the nodes that are farther, cannot reach the target nodes. By propagating the information that way, the MRDP predicted on the target nodes is the same, but it is more efficient from a computational point of view.

4.3.3 Rainfall distribution reconstruction

After useful information reaches the target nodes, the parameters of maximum rainfall distribution must be predicted. This is done by four final dense layers (Figure 4.7), each of which are followed by the activation function LeakyReLU, except from the last one. All the four layers are composed by 64 neurons, except the last one that has a number of neurons equal to the size of the MRDP. These layers are applied node by node, similarly to the feature processing layers (Section 4.3.1). However, for computational reasons, there is no need to apply this layer to all the nodes, it is sufficient to apply these layers only to the target nodes for which the MRDP must be predicted.

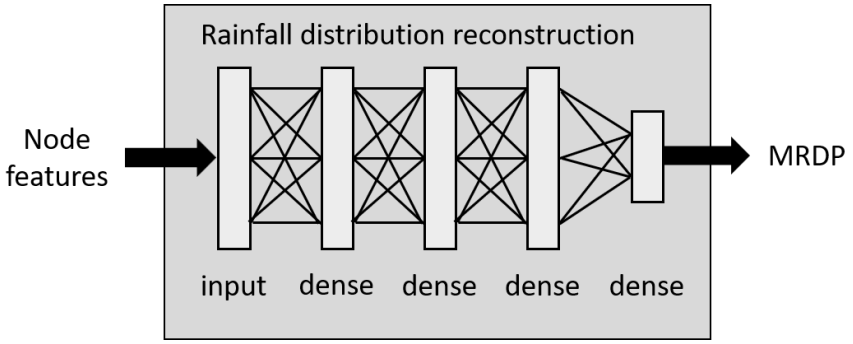


Figure 4.7: Rainfall distribution reconstruction block, which takes as input the node features obtained during graph propagation steps and reconstructs the MRDP.

4.3.4 Dropout and edge drop

Reducing overfitting is often a key part when designing a neural network. In the proposed model, dropout is added after every layer to tackle this problem. However, another possible solution is to use edge drop during training as explained in Section 3.2.5. This solution can be used in replacement of the dropout.

4.4 Learning process

During the training phase, the weights of each layer of the network are changed to improve the results. The measure of improvement is obtained by testing the network on the training set and then score the predictions using a loss function. Mean squared error (MSE) is used in this case. The formula for MSE is the following:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \widetilde{Y}_i)^2$$

where $Y_1, ..., Y_N$ are the predicted MRDP and $\widetilde{Y}_1, ..., \widetilde{Y}_N$ are the real MRDP. In this thesis, the process of optimizing the weights is done an optimizer based on the weights' gradients called Adam [18]. The general idea is to use the gradients of the network's weights to follow the direction for which the loss function decreases.

The magnitude of the gradient can indicate how much the weights should be moved towards the gradient direction. However, during the training phase, the training samples can be weighed differently from each other. This is done to assign less weight to less reliable data. In the case of MRDP, the reliability is given by how well the distribution fits the real data (Kolmogorov-Smirnov or Cramér–von Mises statistics), previously obtained in the preprocessing phase (Section 4.1). Once the reliability value is obtained for each station, each time a station is chosen to be in the batch, the change of weights is weighed by its reliability. This way, the less reliable stations will have less impact on the final weights of the network.

This whole process is further complicated by two main factors. First, as the network works in a transductive way, the nodes for the validation and test set are already present in the training set, even if the values that must be predicted are not known. The second factor is the nature of interpolation problems themselves. In interpolation problems the value that must be predicted (MRDP in this case) is known for some nodes as a feature and must be used to predict the value for the other nodes. This creates multiple type of nodes that must be defined and addressed both during training phase and during validation or test phases.

As explained in Section 4.2.1, there are three types of nodes: known stations, unknown stations and artificial stations. Also, the dataset with real stations is split into training, validation and test sets. What has not yet been defined is the relationship between the type of nodes and the sets of data. Artificial stations are not contained in any set (training, validation or test sets), as they are created artificially. The stations contained in Validation and test sets are considered unknown stations as the rainfall distribution parameters must be predicted for those nodes. Training stations can be considered both known stations or unknown stations, depending on the learning phase, as is described in Sections 4.4.1 and 4.4.2.

4.4.1 Training

For each training step, a random batch of training nodes is selected. Considering that the learning process is transductive, the nodes of the validation and test sets are also already used during training, even if the MRDP is not known for those stations. Therefore, there are four types of nodes: artificial stations, training nodes in the batch, training nodes not in the batch and nodes in validation and test sets. During this phase, the network must learn to predict MRDP for the nodes inside the batch. Therefore, those nodes cannot have the MRDP as an input feature and are considered unknown stations. The training nodes not in the batch are the only nodes for which the MRDP is known. In practice, during the training phase, the features of the nodes are dynamically masked and the masking values is set accordingly. To sum up, the input features for every node is created as follows:

Node type	Environmental features	MRDP	Masking
Artificial stations	Known	Unknown	0
Training nodes in batch	Known	Unknown	0
Training nodes not in batch	Known	Known	1
Validation nodes	Known	Unknown	0
Test nodes	Known	Unknown	0

Table 4.1: Node features composition for different node types during the training phase.

4.4.2 Validation and test

During validation and test phases, all non artificial stations in the training set are considered as known stations, while the nodes in the validation or test sets are considered unknown. the input features for every type of node can be sum up as follows:

Node type	Environmental features	MRDP	Masking
Artificial stations	Known	Unknown	0
Training nodes	Known	Known	1
Validation nodes	Known	Unknown	0
Test nodes	Known	Unknown	0

Table 4.2: Node features composition of different node types during validation and test.

4.5 Ensemble

As described in Section 3.5, ensemble can combine multiple models to get a more accurate prediction. In this thesis, two types of homogeneous ensembles are proposed and combined together to obtain a much better solution. Both ensemble methods are based on a simple average of the multiple models, but differ in the generation phase (explained in 3.5.1). In order to distinguish the two methods, let them be called base ensemble and temporal ensemble.

In the base ensemble, multiple identical neural networks are created as explained in 4.3 and trained independently. The initialization of the networks' weights, as well as some processes such as batch selection in the training phase, are stochastic processes. This leads to a change in the final weights of the network, creating multiple different models which are then combined with a simple average. The pruning phase of the ensemble, defined in 3.5.2, in this case is just skipped.

In the temporal ensemble, as explained in Section 3.5, just one instance of the network is created and trained. For every certain amount of epochs, the state of the network is stored, creating a list of network's copies at different times of the training phase. Once the training ends, the predictions of the last N copies are combined to obtain better results.

These two ensemble methods can then be combined together. First, N

models are created and self-ensembled by using temporal ensemble. Then, the result of the temporal ensembles is combined using the base ensemble. Considering that M sub-models are aggregated using temporal ensemble, the total number of sub-models are $N * M$. In this case, the number of basic ensemble and temporal ensemble models that are combined are:

Ensemble type	Sub-models number
Basic ensemble models (N)	3
Temporal ensemble models (M)	9
Total ensemble models ($N * M$)	27

Table 4.3: Number of ensemble models.

Chapter 5

Methodology grounding and baseline

In this chapter, the algorithms proposed in Chapter 4 are applied to a real dataset containing meteorological stations from the northern part of Italy. First, the dataset is shown and the MRDP representing the data are defined. Then, the environmental features are proposed based on the territory in question. Next, a way of splitting the dataset between training, validation and test sets is shown. Finally, the baselines are presented and a few metrics to compare them with the proposed algorithms are defined.

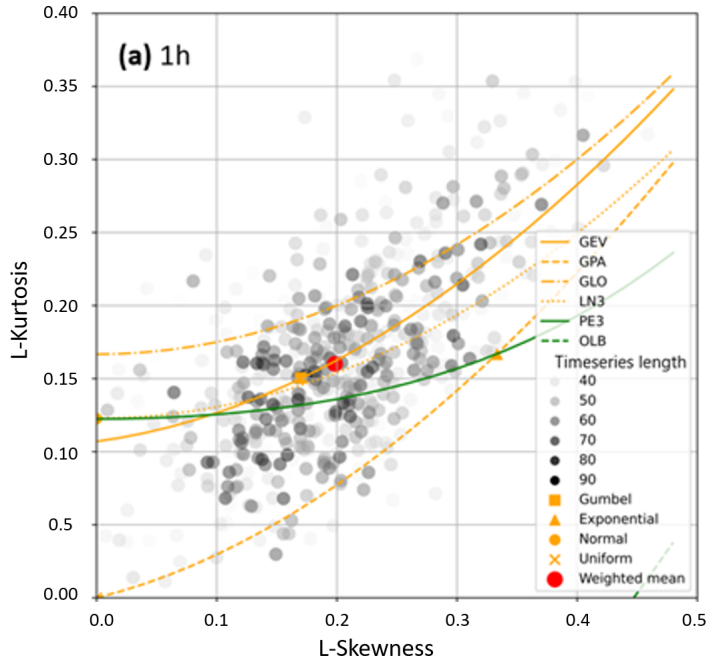
5.1 Northern Italy dataset

In this research, the methods proposed in Chapter 4 have been tested in a practical setting. For this purpose, several stations scattered around the northern part of Italy have been used. For every station, the maximum rainfall depth has been calculated during periods of 1, 3, 6, 12 and 24 consecutive hours, year by year. Most of the stations do not have the data for every year, with some stations having more data than others. This creates a difference in the reliability of maximum rainfall depth for different stations. The information of yearly maximum rainfall, can be summarized over each station through a

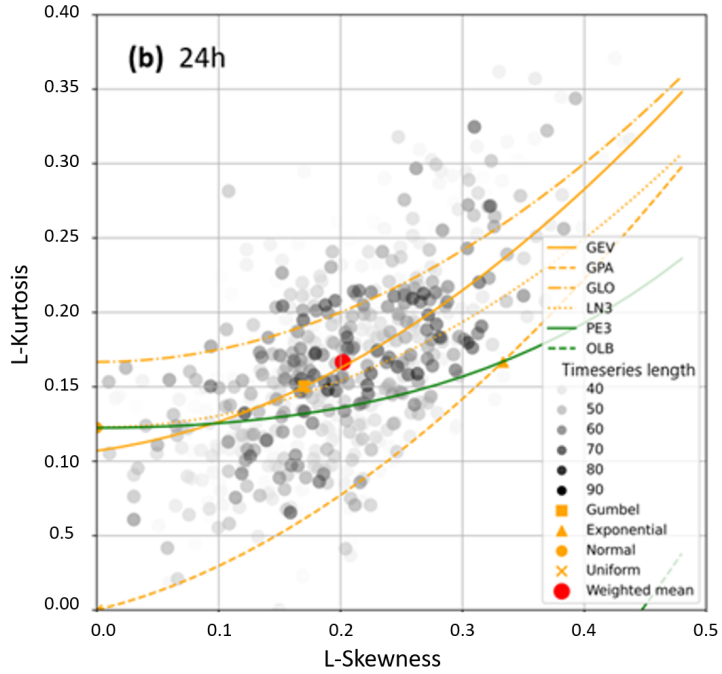
distribution. Once the parameters of the distribution are estimated, it is then possible to make considerations not only about the expected maximum rainfall depth, but also about the probability that the maximum rainfall depth is greater than a certain value during a year long period. Having a good understanding of the probabilities that certain situations occur is the basis for preventing future natural disasters.

According to a study made on the dataset used [27], the Gumbel distribution represents the data with a good approximation. This study uses L-moments to characterize the shape of probability distribution for every station in the dataset. As shown in Figures 5.1a and 5.1b, the weighted average of the station's L-moments is close to the Gumbel's L-moments. Therefore, the Gumbel distribution can be used to represent the data. The Gumbel cumulative distribution function is defined as [33]:

$$F(x, \sigma, \mu) = e^{-e^{\frac{x-\mu}{\sigma}}}$$



(a) L-moments for yearly maximum rainfall depth over 1 hour.



(b) L-moments for yearly maximum rainfall depth over 24 hours.

Figure 5.1

Another possible distribution according to the L-moments shown in Figure 5.1 is the generalized extreme value distribution (GEV). The cumulative density function for the GEV distribution is the following [5]:

$$GEV(x;\epsilon,\sigma,\mu) = e^{-(1+\epsilon*\frac{x-\mu}{\sigma})^{-\frac{1}{\epsilon}}}$$

where $\epsilon \neq 0$ and $\sigma > 0$. In the case that ϵ tends to 0, the GEV distribution corresponds to the Gumbel distribution. This means that, as also shown in Figure 5.1, the Gumbel is a particular case of GEV.

Once the parameters of GEV and Gumbel distributions are fit on the real data, the result is shown in Figures 5.2, 5.3 and 5.4. In Figures 5.3 and 5.4 the parameters σ and μ of GEV and Gumbel distributions are compared. Even if the values are scaled differently, the parameters are distributed similarly over the territory. Also, according to the map shown in Figure 5.2, the parameter ϵ is close to 0, except for some outliers. The fact that for most of the stations the parameter ϵ is close to 0 means that the optimal distribution shape is similar to the Gumbel. Moreover, the presence of few outliers makes it more difficult for any model (both the baselines and the neural network based approaches) to correctly predict the parameters. To summarize, GEV distribution has more parameters which are more difficult to predict while for most of the stations it coincides with the Gumbel. Because of that, Gumbel distribution is a better fit to represent the yearly maximum rainfall of the dataset used.

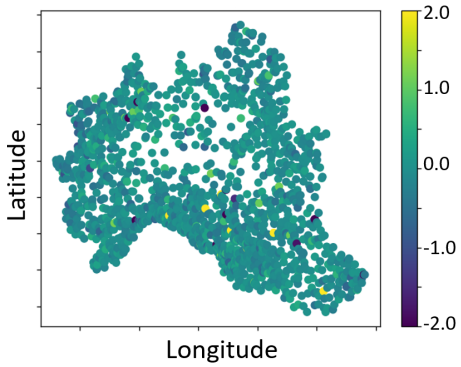
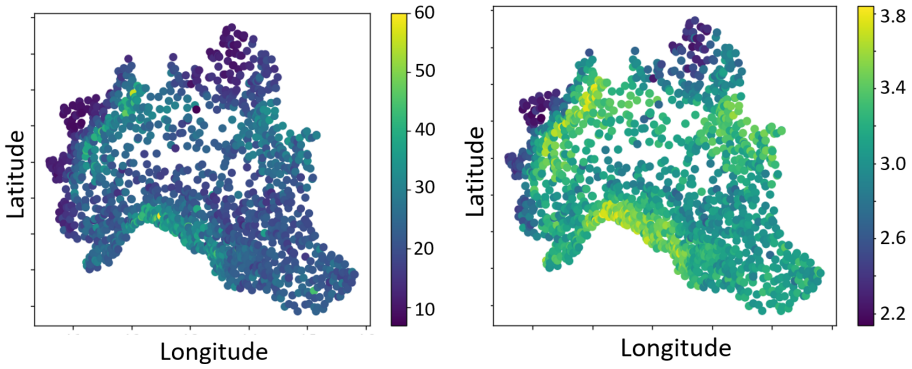
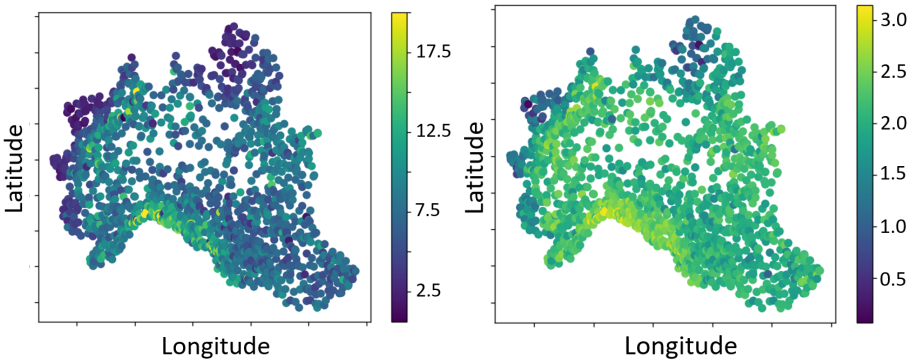


Figure 5.2: GEV parameter ϵ for the stations over northern Italy.



(a) GEV parameter μ. (b) Gumbel parameter μ.

Figure 5.3: Parameters μ for GEV and Gumbel distributions for the stations in northern Italy.



(a) GEV parameter σ. (b) Gumbel parameter σ.

Figure 5.4: Parameters σ for GEV and Gumbel distributions for the stations in northern Italy.

In addition to MRDP, additional information about the territory is needed to correctly predict the MRDP in the unknown stations. Such features can be obtained not only on the stations, but also between them where the real stations are not present, as suggested in Section 4.1.

According to the literature, there is no agreement on the best possible features that must be used for similar problems. Also, some of those features strictly depend on the geographical area where MRDP must be predicted. However, some studies [29, 27] proved that their set of features used for similar problems and geographical areas can lead to good results. In this thesis, a subset of those features is used. Moreover, the features have been adapted

according to the dataset used. The complete list of the environmental features used is presented in Table 5.1:

Name	Description
mean altitude	mean altitude around the target position
std altitude	standard deviation of the altitude around the target position
mean slope	mean slope around the position
mean aspect	mean aspect (direction of maximum slope) around the target position
MAP	mean annual precipitation around the target position
mean snow	mean snow precipitation around the target position
std snow	standard deviation of snow precipitation around the target position
distance from Adriatic	minimum distance from the Adriatic coast
mean altitude from Adriatic	mean altitude between the target position and the Adriatic coast
std altitude from Adriatic	standard deviation of the altitude between the target position and the Adriatic coast
distance from Tyrrhenian	minimum distance from the Tyrrhenian coast
mean altitude from Tyrrhenian	mean altitude between the target position and the Tyrrhenian coast
std altitude from Tyrrhenian	standard deviation of the altitude between the target position and the Tyrrhenian coast

Table 5.1: Environmental feature names and descriptions adapted to northern Italy's dataset. Mean altitude, std altitude, mean slope, mean aspect, MAP, mean snow and std snow are all taken within a radius of 1 km around the target position.

Once those features are created, the dataset is split between training, validation and test sets. In this case the test set contains about 5% of the data, the validation set about 10% and the training set the remaining 85%. As the aim of this thesis is to interpolate the MRDP on the areas between the stations, the validation stations are taken completely randomly as shown in Figure 5.5. Similarly to the validation set, the test set is also well distributed around the domain but has been carefully selected to satisfy three criteria:

- At least some of the stations must have a high number of observations and therefore be reliable enough
- The stations must be well distributed over the territory
- The stations must represent different morphoclimatic contexts (such as plains and different mountain ranges)

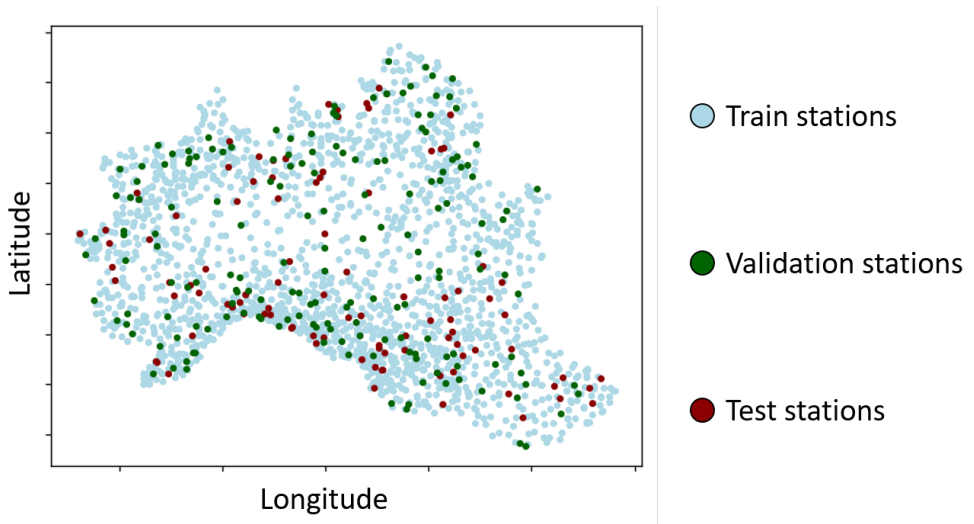


Figure 5.5: Northern Italy's dataset split into training, validation and test sets.

5.2 Baseline

To show the effectiveness of the proposed models and their variations, a selection of the state of the art algorithms is used as baseline. The main aspects

that have been considered to choose the baseline models can be summarized as follows:

- Relevance: only state of the art models known to have high performance in similar problems are used
- Popularity: well known algorithms are preferred as they can give a better understanding of the results
- Simplicity: in order to reduce human error, algorithms that are difficult to calibrate or to apply are excluded. This reduces the possibility of falsifying the results by comparing well optimized algorithms with non optimized ones

Using this reasoning, the selected baseline models are the following:

- Ordinary kriging linear
- Ordinary kriging power
- Universal kriging linear with altitude as external drift
- Gaussian process

Here, there are two versions of ordinary kriging that differ from the variogram model used (explained in Section 3.1.1). In the linear one, the correlation between neighbour stations is assumed to be linear with the distance. In the power variation, the variogram model is a power function where the distance is raised to a certain power that is optimized to fit the data as much as possible. Similarly, universal kriging also used a linear variogram model but also uses the stations altitude to improve the predictions, as explained in Section 3.1.2. When all the known stations are used to predict the target stations, both ordinary kriging and universal kriging perform poorly. Intuitively, far stations which have little to no correlation to target stations affect negatively the predictions. To solve this problem and improve the predictions, for each

target station only the closer stations are interpolated using ordinary kriging or universal kriging. The number of neighbour stations used for interpolation is optimized based on the dataset. Considering the northern Italy dataset, the optimal numbers of neighbours are the following:

Algorithm	Num. of neighbours
Ordinary kriging linear	10
Ordinary kriging power	20
Universal kriging	10

Table 5.2: Optimal number of closer neighbours that are used for interpolation.

In the Gaussian process, as explained in Section 3.1.3, what defines the distribution is the kernel function. In this case, the kernel function is a composition of more sub functions:

$k(x_1, x_2) = WhiteKernel(x_1, x_2) + ConstKernel(x_1, x_2) * RBF(x_1, x_2)$

where:

$$WhiteKernel(x_1, x_2) = \begin{cases} \mu & x_1 = x_2 \\ 0 & x_1 \neq x_2 \end{cases}$$

$ConstKernel(x_1, x_2) = \epsilon \quad \forall x_1, x_2$

$RBF = \exp(\frac{||x_1 - x_2||^2}{2\sigma})$

Where $||...||$ is the euclidean norm.

White kernel is used to consider the fact that there is noise in the data, so the knowledge on the known points is also uncertain. RBF is known as the radial basis function and is multiplied by the constant kernel to weigh it in an automated way. The parameters μ, ϵ and σ are optimized to fit the known data as much as possible. Once these parameters are fitted, the kernel can be used to define the distribution, which is then used to predict the MRDP on the new stations. Differently from Ordinary kriging and universal kriging, Gaussian process performs better when all the stations of the northern Italy dataset are used to create the kernel and then to predict the target stations MRDP.

5.3 Metrics

In order to compare the proposed model with the baselines, several metrics are proposed:

- Mean absolute error (MAE):
- Mean squared error (MSE)
- Root mean squared error (RMSE)

Defined as:

$$MAE(X,Y) = \frac{1}{N}\sum_{i=1}^{N}|x_i - y_i|$$
$$MSE(X,Y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2$$
$$RMSE(X,Y) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2}$$

Chapter 6

Empirical evaluation

In this chapter, the results obtained by the different model variations are compared. In the final test, only the two main variations (Tiny eGAT and Extended eGAT) are tested on the test set. Because Tiny eGAT is computationally less expensive and has similar results to extensive eGAT, the other variations have been tested only on the Tiny eGAT. Also, to avoid choosing the other variations knowing the test set, the tests done to demonstrate the effectiveness of these variations have only been done using the validation set.

6.1 Tiny eGAT and Extended eGAT final test

After assessing the best combination of hyper-parameters and variations, the two main models (Tiny eGAT and Extended eGAT) have been trained and then tested on the test set. The result is then compared with the scores obtained by the baselines in Table 6.1:

	MSE	RMSE	MAE
Tiny eGAT	64.52 ± 1.53	7.34 ± 0.072	4.65 ± 0.042
Extended eGAT	**62.00 ± 0.94**	**7.24 ± 0.048**	**4.54 ± 0.025**
Ordinary kriging linear	75.23	7.73	4.89
Ordinary kriging power	75.47	7.77	4.98
Universal kriging	86.48	8.33	5.17
Gaussian process	86.01	8.23	5.38

Table 6.1: Comparison between the baselines and the proposed algorithms Tiny eGAT and Extended eGAT, when tested on the test set. In the case of Tiny eGAT and Extended eGAT, the metrics represent the average and standard deviation (preceded by the symbol ±) across multiple training and testing iterations.

According to all the considered metrics (MSE, RMSE and MAE), both proposed algorithms, Tiny eGAT and Extended eGAT, perform consistently better than all the baselines, with Extended eGAT performing better. The standard deviation of the metrics between different runs is much smaller than the metrics difference between the proposed algorithms (Tiny eGAT and Extended eGAT) and the baselines, ensuring consistent results across different tests. Between the baselines, ordinary kriging linear is the best baseline according to all the metrics. Universal kriging with altitude as external drift performs poorly. One possible motivation is that there is no strong direct correlation between the MRDP and the altitude.

6.1.1 Best variation selection

While the test shown in section 6.1 has been done with the best variations applied to Tiny eGAT and Extended eGAT, the selection of those variations needs to be justified. In order to prove the effectiveness of those variations,

more tests have been done on the validation set. The detailed reasoning behind the variations selection will be described in the next sections. The list of variations used can be summarized as follows:

	Tiny eGAT	Extended eGAT
Layer jumping	Residuals	Jumping knowledge
Ensemble	Base ens. of temporal ens.	Base ens. of temporal ens.
Sample weights	Cramér von Mises	Cramér von Mises
Overfitting	Edge drop	Edge drop

Table 6.2: Variations selection.

In order to contextualize the next tests that will be shown, it is useful to compare the validation and test sets. While the baselines performances on the test set have already been shown (Table 6.1), their performances on the validation set can be seen in Table 6.3. It is important to consider that both the baselines, Tiny eGAT and Extended eGAT perform better on the validation set. As the difference in performance is consistent with all techniques tested, it is plausible that the comparison of performances for different variations seen on the validation set remains more or less invariant in the test set.

	MSE	RMSE	MAE
Tiny eGAT	46.61	6.21	4.16
Extended eGAT	45.51	6.13	4.07
Ordinary kriging linear	62.05	6.94	4.61
Ordinary kriging power	59.78	6.83	4.51
Universal kriging	65.60	7.14	4.76
Gaussian process	59.02	6.77	4.58

Table 6.3: Comparison between the baselines on the validation set.

6.2 Deeper network and jumping knowledge

Graph neural networks are limited on how far the information can travel in the graph. If the information is not aggregated from far enough nodes, predicting MRDP correctly becomes difficult. Also, how far the information must be propagated strictly depends on the dataset. Therefore, it is important to adapt the network to propagate the information far enough. This can be done in two ways: by increasing the number of GNN layers or by creating edges that connect nodes that are farther.

The creation of long distance edges increases the number of neighbours for every node and by consequence the number of nodes that are aggregated in each GNN layer. When too many nodes are aggregated, it is difficult to extract all the useful information from neighbour nodes. Therefore, it could become difficult for the network to correctly model the relationships between the nodes. Adding more GNN layers, on the other hand, makes it more difficult for the optimizer to find the best combination of network weights. However, this problem can be solved by using the jumping knowledge technique.

When tested on the northern Italy dataset, the Tiny eGAT with residuals performs better than Extended eGAT with residuals as shown is Figures 6.1 and 6.2. However, when tested with the jumping knowledge variation instead of residuals, both Tiny eGAT and Extended eGAT perform similarly according to MSE or RMSE (Figure 6.1) and better when compared with residuals, while Extended eGAT performs slightly better only when compared using MAE metric (Figure 6.2). This means that while the performances are always improved when jumping knowledge is used, Extended eGAT obtains more advantage from using it compared to Tiny eGAT.

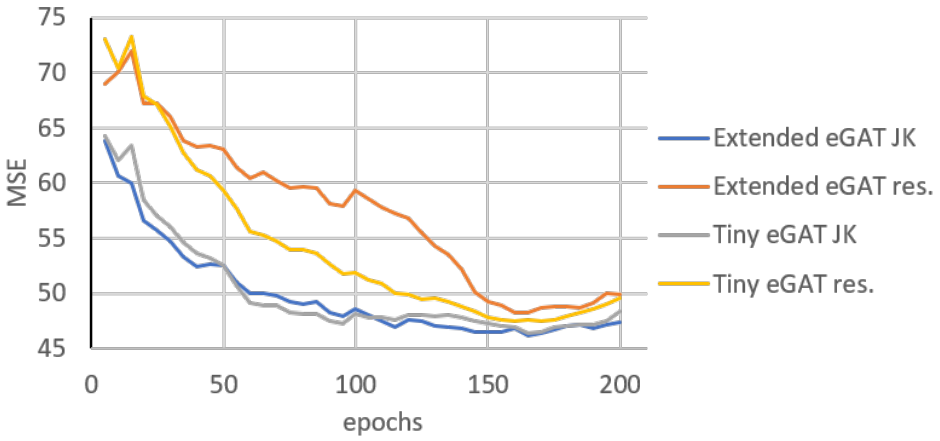


Figure 6.1: Comparison of MSE between Tiny eGAT or Extended eGAT when using jumping knowledge (JK) or residuals (res).

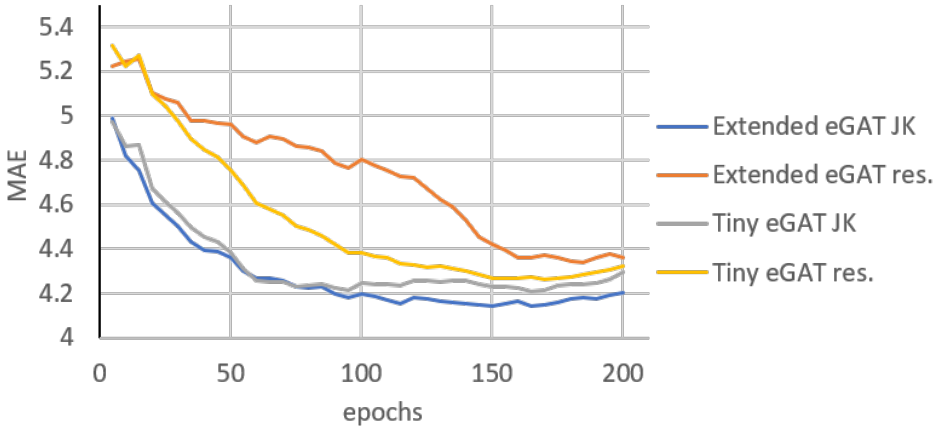


Figure 6.2: Comparison of MAE between Tiny eGAT or Extended eGAT when using jumping knowledge (JK) or residuals (res).

6.2.1 Resource consumption

The jumping knowledge variation considerably increases the size of the network. This leads to an increase in the resource consumption. Similarly, the same thing happens when more GP blocks (described in Section 4.3.2) are added. This suggests two main variations of the network: Tiny eGAT with residuals which is lighter but worse at predicting MRDP and Extended eGAT

with jumping knowledge which is heavier but better at predicting MRDP. A third valid variation is Tiny eGAT with jumping knowledge. However, this last combination is not further explored in this thesis.

6.3 Ensemble evaluation

In this thesis two models of ensemble are proposed (Section 4.5): base ensemble and temporal ensemble. Therefore, both ensembles have been tested and compared when used individually and combined together. These tests demonstrate that the use of both base ensemble and temporal ensemble not only improves the results but also makes the results more stable over different training and testing iterations or different stopping epochs. Then, by comparing the predictions of different sub models, a measure of prediction reliability is obtained and compared with the actual results.

6.3.1 Base ensemble

By applying the base ensemble, combining the predictions of three different instances of the Tiny eGAT, it is possible to decrease the prediction error as shown in Figure 6.3. Also, Figure 6.4 shows that the prediction error for the base ensemble is lower even when compared with the average of the sub-models prediction error. Moreover, from Figure 6.4 it is possible to deduce that the base ensemble prediction error is more or less proportional to the average of the sub-models prediction error. This leads to the conclusion that even if the performances of the sub-models vary one from the other, by aggregating multiple sub-models, the results are more stable over multiple training and testing iterations and less dependent on chance.

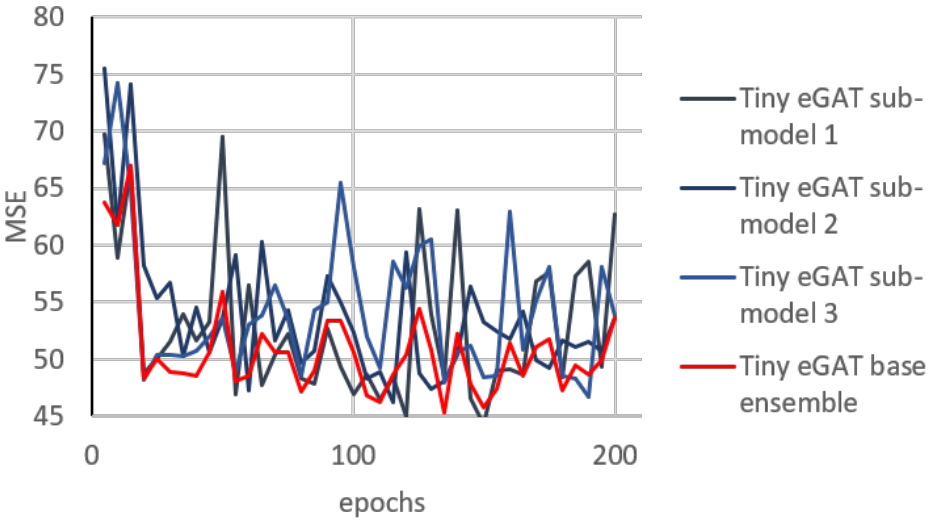


Figure 6.3: Prediction error of three Tiny eGAT models (sub-models 1, 2 and 3) compared with the base ensemble that combines the predictions of those models together.

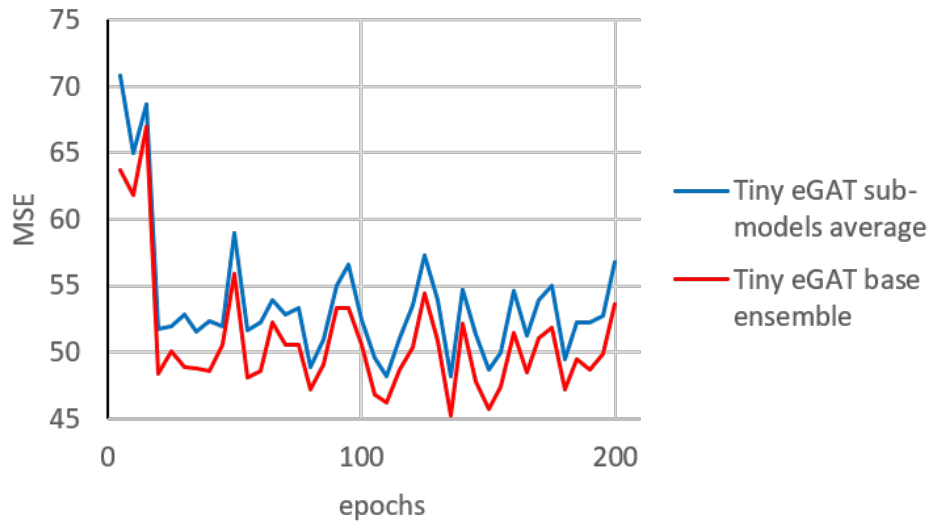


Figure 6.4: Base ensemble prediction error compared with the average of the three sub-models prediction error.

6.3.2 Temporal ensemble evaluation

By applying the temporal ensemble to the network, it is possible to achieve two important improvements as shown in Figure 6.5: better performances and more stability in the results. It is important to notice that the prediction error of the temporal ensemble model oscillates less over time, therefore, the prediction error depends less on the stopping epoch. This is important as it influences the performances when tested on different datasets such as the test set. Usually, the validation set is assumed to be similar to the test set and the stopping epoch is chosen based on the validation set. However, the optimal stopping epoch for the test set can differ from the one for the validation set. By decreasing the oscillation in the prediction error over different epochs, the consequences of choosing a sub-optimal stopping epoch are also reduced.

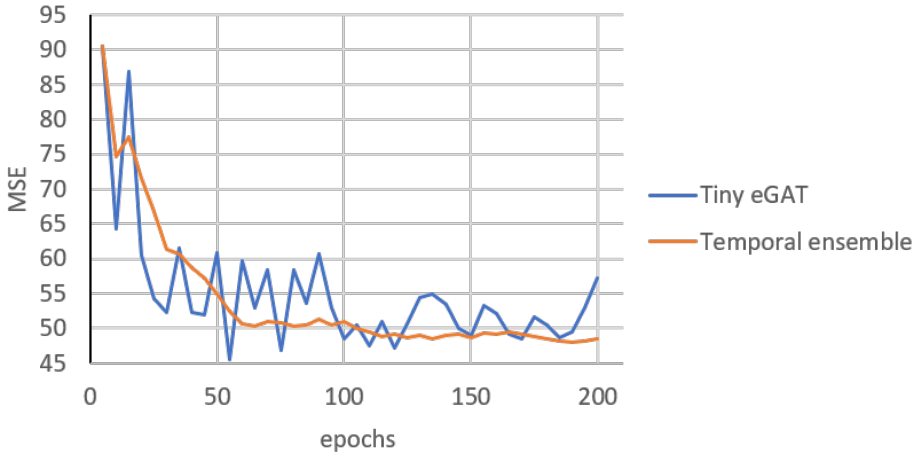


Figure 6.5: Tiny eGAT prediction error on the validation set compared with the same model after applying temporal ensemble using 9 sub-models spaced by 5 epochs each.

6.3.3 Ensemble of ensembles

Previous tests demonstrated the advantages of using base ensemble or temporal ensemble alone. In particular, other than improving the overall results, the base ensemble reduces the variation of the results over multiple tests while the

temporal ensemble reduces the oscillation of the results over different epochs during training.

By applying the base ensemble to multiple temporal ensembles (ensemble of ensembles), each sub-model (temporal ensembles) already benefits from the reduction of oscillation over different epochs. When the sub-models are then combined using the base ensemble, as shown in Figure 6.6, the first property (reduction in the oscillation over different epochs) is maintained. Other than that, the similarity between the prediction error of the ensemble of ensembles and the average of the sub-models prediction error (Figure 6.7) is analogue to the one described in Section 6.3.1. Therefore there is a reduction in the variation between the results of different training and testing iterations. To sum up, by applying the base ensemble to temporal ensembles the results are improved and the advantages of both base ensemble and temporal ensemble are maintained, obtaining an ensemble which is better than both base ensemble and temporal ensemble individually.

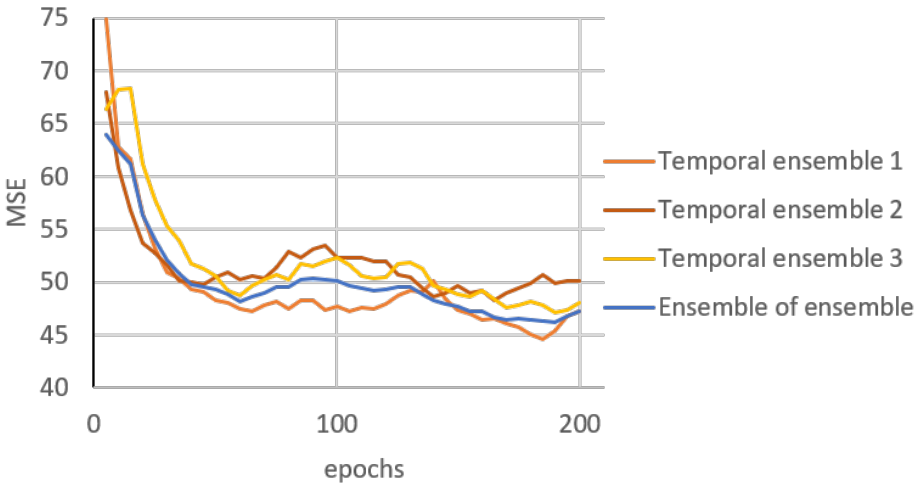


Figure 6.6: Base ensemble of temporal ensembles (ensemble of ensembles) compared with its sub-models (temporal ensembles).

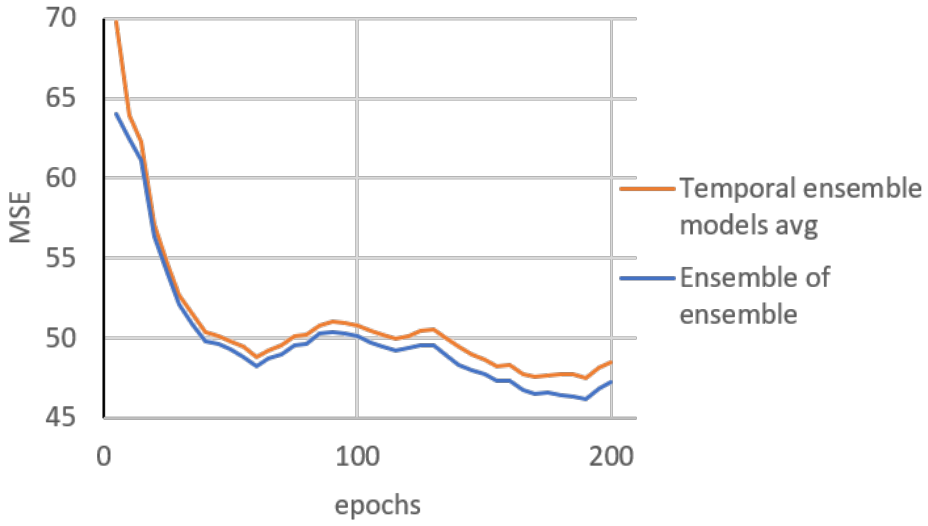


Figure 6.7: Base ensemble of temporal ensembles (ensemble of ensembles) compared with the average of its sub-models (temporal ensemble models avg).

6.3.4 Reliability measure

Given the prediction of MRDP on a certain station, the prediction reliability estimates how much the predictions are likely to be accurate. To show that reliability measure is coherent with the results, a relationship must be found between the prediction error and the reliability measure.

By using the ensemble algorithm, many sub models are created. If all the models agree on the same MRDP for a certain station, it is plausible that the station is easier to predict. If the station is easier to predict, the prediction is more reliable. Therefore, a measure of sub model accordance can be used as a measure of reliability. By using standard deviation between the predictions of different sub-models as measure of accordance, the results shown in Figure 6.8 are obtained. In this test, standard deviation (std) and MSE are first measured for each parameter in MRDP and then averaged and compared. Similar results are obtained when parameters of MRDP are considered one at the time. These results show some correlation between sub-model accordance (std) and ensemble MSE, meaning that when the sub-models disagree, the ensemble

model is more likely to fail. This information can be used, together with the predictions, to have a better understanding about the distribution of maximum rainfall over the territory.

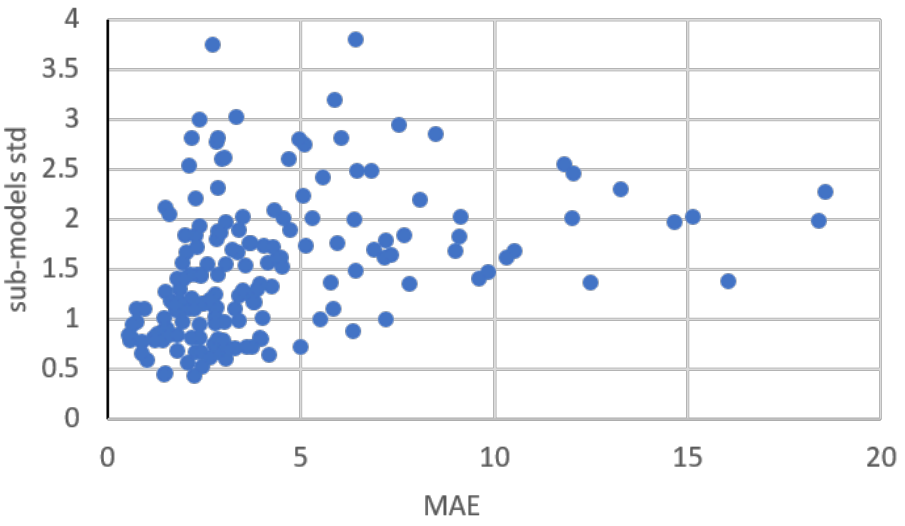


Figure 6.8: Standard deviation (std) between the predictions of different ensemble sub-models compared with ensemble MSE. In this case both std and MSE are first calculated for each parameter in MRDP and then averaged.

6.4 Sample weights

The MRDP that must be interpolated is an estimation and therefore prone to errors and misrepresentation. Using unreliable MRDP can lead to the creation of a model that misrepresents the training dataset. However, this can be solved by weighing the training samples as explained in Section 4.4. By observing the graph in Figure 6.9, it is unclear whether Cramér von Mises or Kolmogorov-Smirnov is better to weigh the samples. However, both cases decrease the MSE, compared to not weighing the samples. By considering RMSE or MAE similar results are obtained.

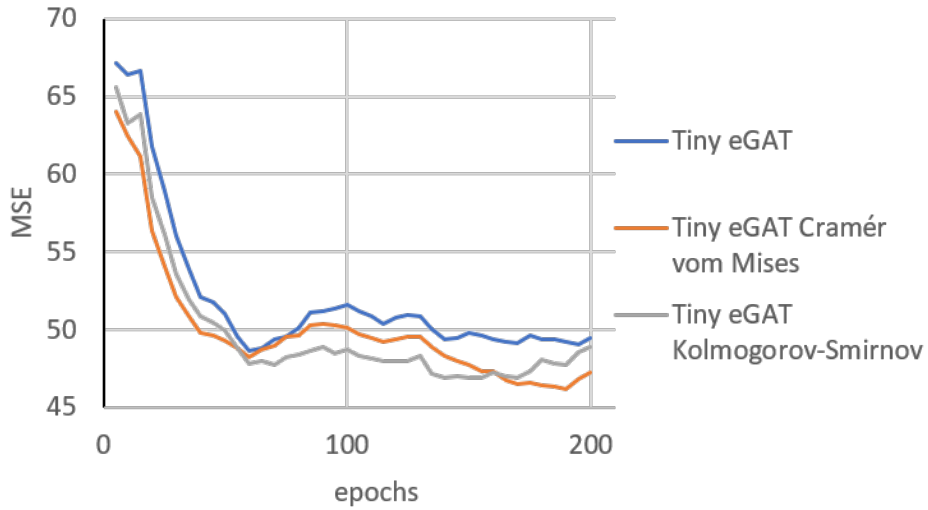


Figure 6.9: Tiny eGAT without any sample weighing compared with Tiny eGAT with the training samples weighed based on Cramér von Mises or Kolmogorov-Smirnov.

6.5 Overfitting, dropout and edge drop

Reaching a compromise between generalizing too much and generalizing too little is fundamental to obtain good results. In this thesis, two possible solutions are proposed to help the network generalizing: dropout and edge drop. The first step is understanding whether the network without dropout or edge drop is overfitting or not. As shown in Figures 6.10 and 6.11, the difference in the performance of the network when tested on training set and validation set is relevant. On the other hand, the results on the validation set are already good compared with the baselines. This suggests that there is some overfitting, even if limited, and improving the results could be possible by reducing it.

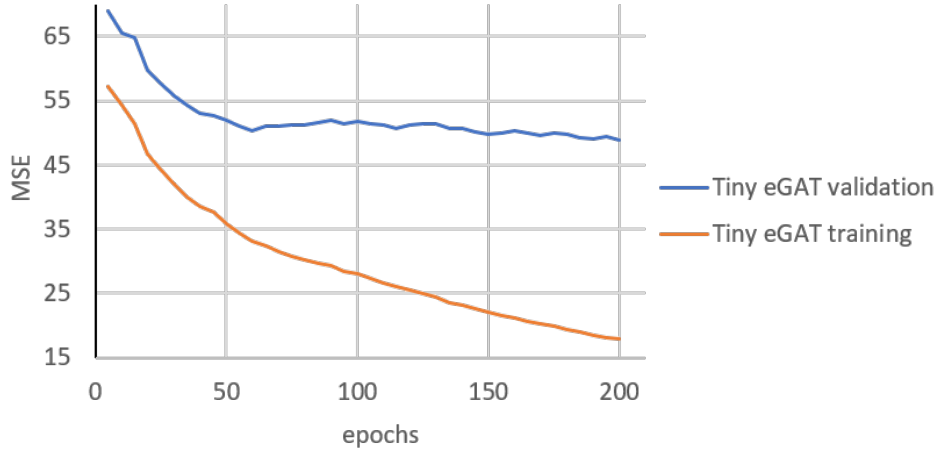


Figure 6.10: Tiny eGAT MSE on validation set without the addition of dropout or edge drop.

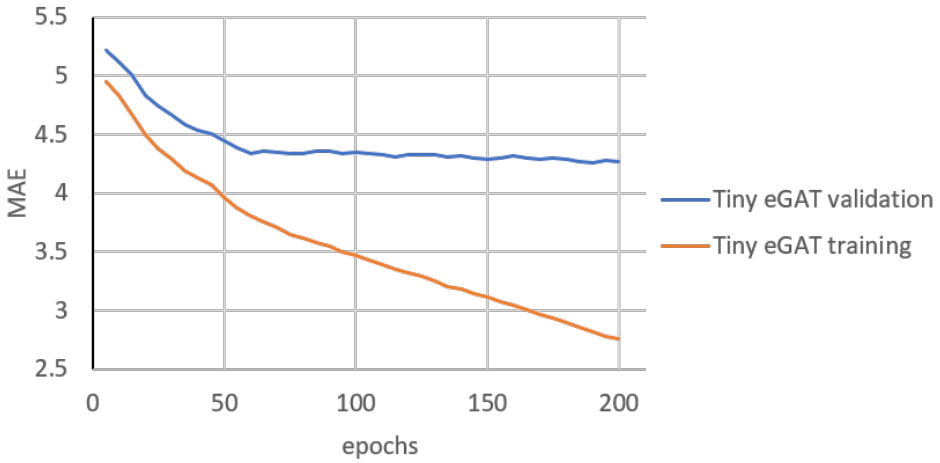


Figure 6.11: Tiny eGAT MAE on validation set without the addition of dropout or edge drop.

Three variations of Tiny eGAT have been compared:

- Dropout with 0.1 probability of removing connections
- Edge drop with 0.5 probability of removing edges
- Without dropout or edge drop

Figures 6.12 and 6.13 show the prediction error of the different variations on the training and validation sets. Both dropout and edge drop variations cause a reduction in the difference between training and validation error, as also confirmed in Figure 6.14. This proves that both variations help generalizing, with dropout generalizing more. However, reducing overfitting is pointless if there is no improvement in the results. By also considering the results on the validation set (Figure 6.13), while dropout reduces overfitting, it does not improve the results. By applying edge drop variation, while the reduction of difference between training and validation MSE is smaller, better results are obtained when tested on the validation set. Therefore, edge drop is a better option than dropout.

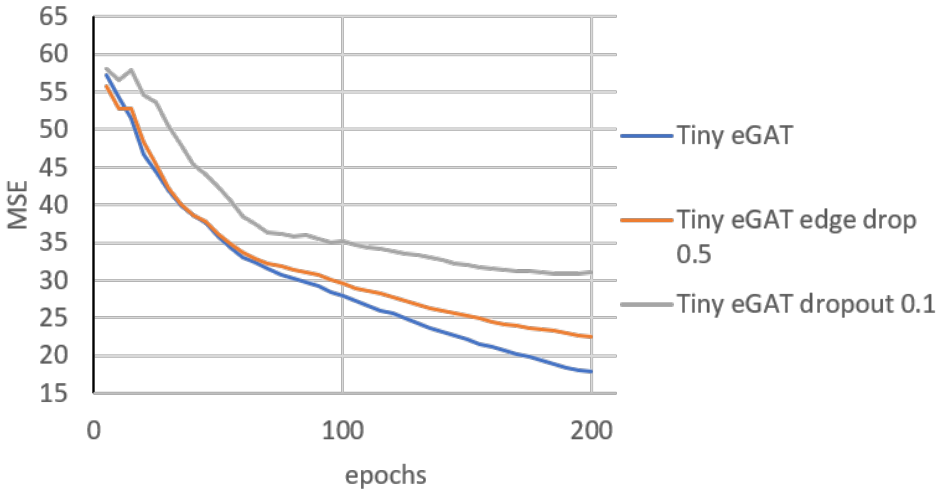


Figure 6.12: Tiny eGAT prediction error on training set with or without dropout or edge drop. In this case, the probabilities of dropping the connection or edge are 0.1 and 0.5.

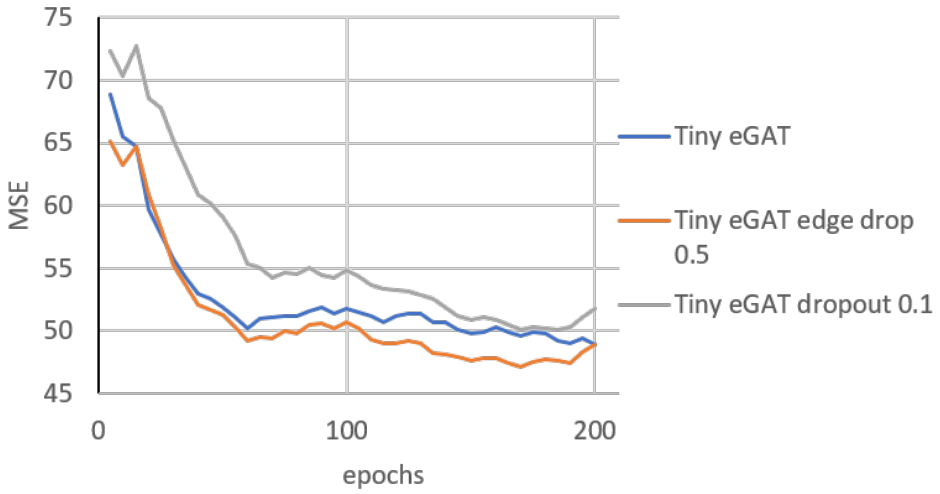


Figure 6.13: Tiny eGAT prediction error on validation set with or without dropout or edge drop. In this case, the probabilities of dropping the connection or edge are 0.1 and 0.5.

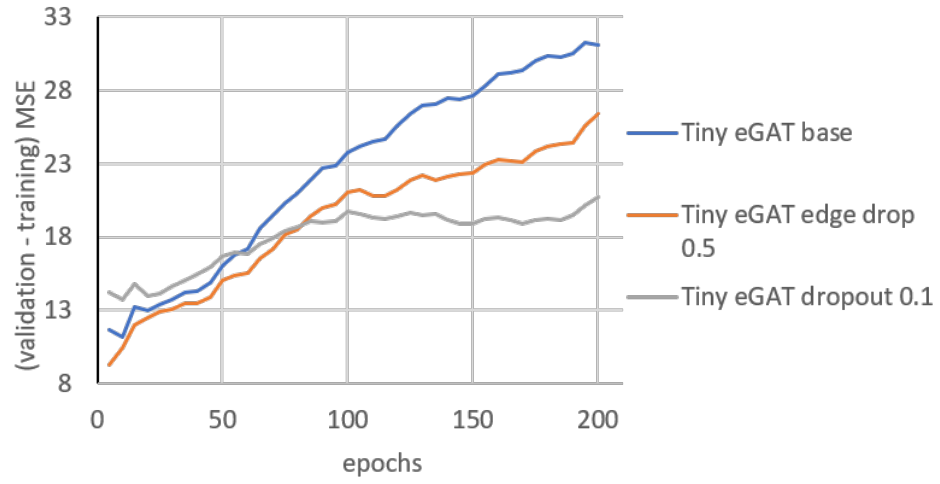


Figure 6.14: Prediction error difference between training and validation sets of Tiny eGAT variations. In this case, the probabilities of dropping the connection or edge are 0.1 and 0.5.

Chapter 7

Conclusions

In this thesis, the problem of maximum rainfall depth distribution parameters interpolation has been tackled through the introduction of two models: Tiny eGAT and Extended eGAT, both based on edge graph attention layers. While Tiny eGAT is computationally lighter, Extended eGAT is designed to be more adaptive through the incorporation of additional GNN layers. Various network variations have been proposed and tested on the northern Italy dataset. Following that, Tiny eGAT and Extended eGAT were compared against baseline algorithms, including ordinary kriging, universal kriging, and Gaussian process. In the evaluation on the northern Italy dataset, both Tiny eGAT and Extended eGAT consistently outperformed the baseline algorithms. However, when compared to each other, both models yielded similar results, with Extended eGAT performing better.

While Tiny eGAT demonstrated satisfactory performance with residuals, Extended eGAT necessitated jumping knowledge to reach the same results, thereby increasing training time. To enhance results, base ensemble and temporal ensemble were tested individually and in combination. Both ensemble types proved effective in improving results, with the base ensemble reducing variability between different tests and the temporal ensemble reducing variability over time. The combined use of both ensembles not only further improved the performances but also retained the advantages provided by each

type of ensemble. Additionally, the use of ensembles allowed for the possibility of utilizing differences in sub-model outputs to assess prediction reliability. Furthermore, an estimation of station reliability was employed to weigh training samples, mitigating the impact of noisy stations on the results. Finally, the issue of overfitting was addressed by proposing two potential solutions: dropout and edge drop. Between the two, edge drop demonstrated significantly better generalization capabilities.

The two variations Tiny eGAT and Extended eGAT have been effectively optimized for predicting MRDP on the northern Italy dataset. Nevertheless, with certain modifications, they could also be applied to other datasets. These adjustments should include changes in the environmental features and the number of GP blocks. The optimal number of GP blocks may vary across different datasets and even within distinct areas of the same dataset. In such cases, an ideal solution would involve a dynamic network where the number of GP blocks adapts based on the specific area. Jumping knowledge could facilitate this adaptation by learning how and when to skip entire portions of the network. However, whether jumping knowledge can achieve similar results on other datasets remains an area for further exploration.

Bibliography

[1] P. Ahrendt. The multivariate gaussian probability distribution. *Technical University of Denmark, Tech. Rep*:203, 2005.

[2] D. Anderson and G. McNeill. Artificial neural networks technology. *Kaman Sciences Corporation*, 258(6):1–83, 1992.

[3] T. W. Anderson and D. A. Darling. A test of goodness of fit. *Journal of the American statistical association*, 49(268):765–769, 1954.

[4] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[5] T. G. Bali. The generalized extreme value distribution. *Economics letters*, 79(3):423–427, 2003.

[6] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[7] P.-S. Chu, X. Zhao, Y. Ruan, and M. Grubbs. Extreme rainfall events in the hawaiian islands. *Journal of Applied Meteorology and Climatology*, 48(3):502–516, 2009.

[8] K. K. Evans, T. S. Horowitz, P. Howe, R. Pedersini, E. Reijnen, Y. Pinto, Y. Kuzmova, and J. M. Wolfe. Visual attention. *Wiley Interdisciplinary Reviews: Cognitive Science*, 2(5):503–514, 2011.

[9] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[10] P. Goovaerts. *Geostatistics for natural resources evaluation*. Applied Geostatistics, 1997.

[11] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Volume 2, pages 729–734. IEEE, 2005.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[14] K.-l. Hsu, X. Gao, S. Sorooshian, and H. V. Gupta. Precipitation estimation from remotely sensed information using artificial neural networks. *Journal of Applied Meteorology and Climatology*, 36(9):1176–1190, 1997.

[15] M. F. Hutchinson. Interpolating mean rainfall using thin plate smoothing splines. *International journal of geographical information systems*, 9(4):385–403, 1995.

[16] A. Justel, D. Peña, and R. Zamar. A multivariate kolmogorov-smirnov test of goodness of fit. *Statistics & probability letters*, 35(3):251–259, 1997.

[17] E. R. Kandel, J. H. Schwartz, T. M. Jessell, S. Siegelbaum, A. J. Hudspeth, S. Mack, et al. *Principles of neural science*, volume 4. McGraw-hill New York, 2000.

[18] D. P. Kingma and J. Ba. Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[19] A. Krenker, J. Bešter, and A. Kos. Introduction to the artificial neural networks. *Artificial Neural Networks: Methodological Advances and Biomedical Applications. InTech*:1–18, 2011.

[20] R. J. Kuligowski and A. P. Barros. Using artificial neural networks to estimate missing rainfall data 1. *JAWRA Journal of the American Water Resources Association*, 34(6):1437–1447, 1998.

[21] M. Kumari, A. Basistha, O. Bakimchandra, and C. Singh. Comparison of spatial interpolation methods for mapping rainfall in indian himalayas of uttarakhand region. In *Geostatistical and Geospatial Approaches for the Characterization of Natural Resources in the Environment: Challenges, Processes and Strategies*, pages 159–168. Springer, 2016.

[22] S. Laine and T. Aila. Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242*, 2016.

[23] J. Li, Y. Shen, L. Chen, and C. W. W. Ng. Rainfall spatial interpolation with graph neural networks. In *International Conference on Database Systems for Advanced Applications*, pages 175–191. Springer, 2023.

[24] J. Li, Y. Shen, L. Chen, and C. W. W. Ng. Ssin: self-supervised learning for rainfall spatial interpolation. *Proceedings of the ACM on Management of Data*, 1(2):1–21, 2023.

[25] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis. Dying relu and initialization: theory and numerical examples. *arXiv preprint arXiv:1903.06733*, 2019.

[26] A. L. Maas, A. Y. Hannun, A. Y. Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30 of number 1, page 3. Atlanta, GA, 2013.

[27] A. Magnini, M. Lombardi, T. B. Ouarda, and A. Castellarin. Ai-driven morphoclimatic regional frequency modelling of sub-daily rainfall-extremes. *Available at SSRN 4489548*.

[28] A. Mair and A. Fares. Comparison of rainfall interpolation methods in a mountainous region of a tropical island. *Journal of hydrologic engineering*, 16(4):371–383, 2011.

[29] P. Mazzoglio, I. Butera, M. Alvioli, and P. Claps. The role of morphology in the spatial distribution of short-duration rainfall extremes in italy. *Hydrology and Earth System Sciences*, 26(6):1659–1672, 2022.

[30] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[31] J. Mendes-Moreira, C. Soares, A. M. Jorge, and J. F. D. Sousa. Ensemble approaches for regression: a survey. *Acm computing surveys (csur)*, 45(1):1–40, 2012.

[32] T. Monninger, J. Schmidt, J. Rupprecht, D. Raba, J. Jordan, D. Frank, S. Staab, and K. Dietmayer. Scene: reasoning about traffic scenes using heterogeneous graph neural networks. *IEEE Robotics and Automation Letters*, 8(3):1531–1538, 2023.

[33] S. Nadarajah, S. Kotz, et al. The beta gumbel distribution. *Mathematical Problems in engineering*, 2004:323–332, 2004.

[34] X. Peng, Q. Li, and J. Jing. Cngat: a graph neural network model for radar quantitative precipitation estimation. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–14, 2021.

[35] Y. Rong, W. Huang, T. Xu, and J. Huang. Dropedge: towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

[36] E. Schulz, M. Speekenbrink, and A. Krause. A tutorial on gaussian process regression: modelling, exploring, and exploiting functions. *Journal of Mathematical Psychology*, 85:1–16, 2018.

[37] S. Sharma, S. Sharma, and A. Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017.

[38] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[39] W. Taesombat and N. Sriwongsitanon. Areal rainfall estimation using spatial interpolation techniques. *Science Asia*, 35(3):268–275, 2009.

[40] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[41] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017.

[42] Z. Xiang and I. Demir. High-resolution rainfall-runoff modeling using graph neural network. *arXiv preprint arXiv:2110.10833*, 2021.

[43] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*, pages 5453–5462. PMLR, 2018.

[44] X. Ying. An overview of overfitting and its solutions. In *Journal of physics: Conference series*, volume 1168, page 022022. IOP Publishing, 2019.

[45] T. Zhang, S.-Y. Liew, and H.-F. Ng. A survey on using spatio-temporal networks for rainfall prediction. In *2023 4th International Conference*

on Artificial Intelligence and Data Sciences (AiDAS), pages 36–41. IEEE, 2023.